Piotr Luszczek

CUDA's Asynchronous and Non-Blocking Techniques

Virtual Memory: Page-locked / Pinned Pages

- C library's malloc() is used to allocate memory in the host
  - But malloc() only allocates pageable host memory pages
- cudaHostAlloc() allocates pages from page-locked pool:

```
cudaHostAlloc( (void**)&a, size * sizeof(*a),
cuda HostAllocDefault )
cudaFreeHost ( a )
```

- Page-locked memory pages stay in the physical memory and OS does not evict them onto disk (as is the case for regular virtual memory pages)

Paged Memory Properties

- Copying content of pageable memory to GPU
  - CPU copies data from pageable memory to a page-locked memory
  - GPU uses direct memory access (DMA) to move the data to/from the host's page-locked memory (copy operation is done twice when using C's malloc allocator)
- When using page-locked memory (from CudaHostAlloc), the first copy is skipped
- Page-locked memory allows faster copies but uses physical memory pages and cannot be swapped to disk
  - Total number of pinned pages is limited and the system may run out of memory
  - MPI uses pinned pages for communicating over interconnect

Introduction to CUDA Streams

- Streams introduce task-based parallelism to CUDA codes
  - Kernel launches and CPU-GPU communication are the tasks
- Plays an important role in adding overlap to CUDA-accelerated the applications
- A CUDA Stream represents a queue-like functionality for GPU operations that will be executed in the order of insertions:
  - The order in which the operations are added to a stream specifies the order in which they will be executed
- GPU hardware helps in scheduling operations enqueued in streams
  - Fermi required clever ordering to achieve overlap
  - Kepler introduced HyperQ: 4 hardware queues to execute stream operations and ease the programming effort

Using One CUDA Stream

- First, check if the device supports the ‘device overlap’ property
- Call CudaGetDeviceProperties() to check for device overlap:

```
cudaDeviceProp prop;
int whichDevice;
cudaGetDevice( &devID );
cudaGetDeviceProperties( &prop, devID );
if (0 == prop.deviceOverlap) {
  fprintf(stderr, "No handle overlap support");
}
```

- GPU with overlap capability (Fermi and above) may execute a kernel while performing a memory copy between to/from host

Using One CUDA Stream – contd.

- cudaStreamCreate() creates a new stream:

```
// initialize the stream and create the stream
cudaStream_t  stream;
cudaStreamCreate( &stream );
```

- Allocate the memory on the host and GPU

```
//pagelocked memory at GPU
cudaMalloc( (void**)&dev_a, N*sizeof(int) ) );
// allocate page-locked memory
cudaHostAlloc( (void**)&host_a,
FULL_DATA_SIZE*sizeof(int),
cudaHostAllocDefault );
```

- CudaMemcpyAsync() copies data to/from CPU to GPU. The copy continues after the call returns – completion happens later.

```
cudaMemcpyAsync( dev_a, host_a+i, N*sizeof(int),
cudaMemcpyHostToDevice, stream );
```

Using One CUDA Stream – prolog

- Sample kernel launch

```
kernel<<<N/256,256,0,stream>>>(dev_a,dev_b,dev_c);
```

- copy back data from device to locked memory

```
cudaMemcpyAsync( host_c+i, dev_c, N*sizeof(int),
cudaMemcpyDeviceToHost, stream);
```

- Stream synchronization waits for the stream finish all operations

```
cudaStreamSynchronize(stream);
```

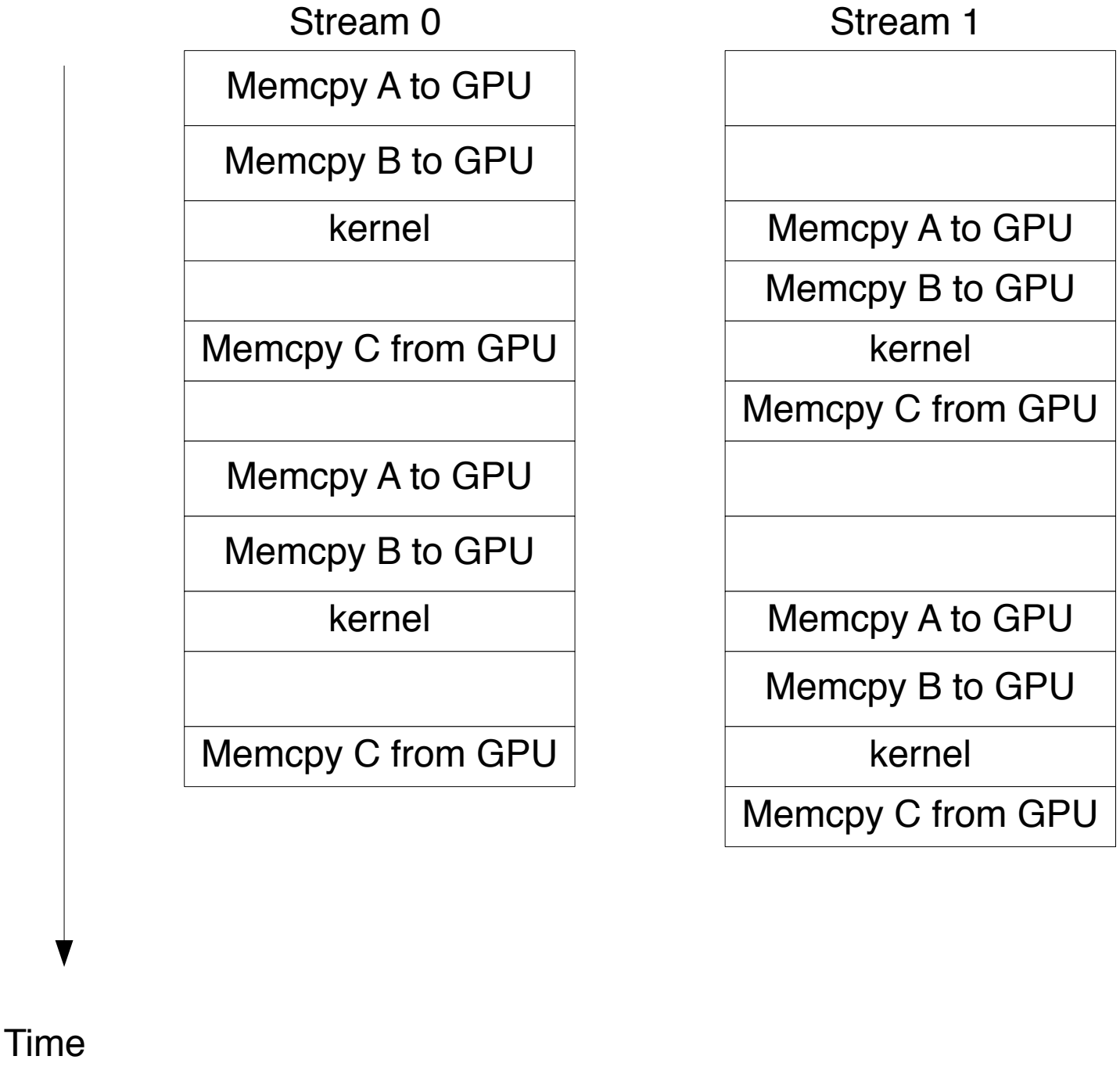
- Free the memory allocated and destroy the stream after synchronization:

```
cudaFreeHost(host_a);

cudaFree(dev_a);

cudaStreamDestroy(stream);
```

Using More than one Streams

- Kernel execution and memory copies can be performed concurrently as long as:
  - they are in multiple streams and,
  - hardware supports
- Some GPU architectures support concurrent memory copies if they are in opposite directions
- The concurrency when multiple streams are helps with:
  - Improving performance
  - Using PCIexpress bus more efficiently

Execution Time Line for 2 Streams



GPU Work Scheduling

- Hardware has no notion of streams
- Hardware has separate queues (engines) to perform memory copies and to execute kernels
- The commands in the queue are like dynamically scheduled task