GPU Teaching Kit

Accelerated Computing

Module 14 – Efficient Host-Device Data Transfer

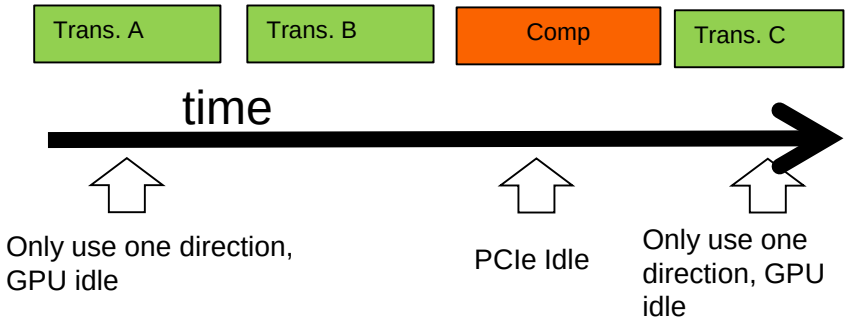
Lecture 14.2 - Task Parallelism in CUDA

Objective

– To learn task parallelism in CUDA
  – CUDA Streams

Serialized Data Transfer and Computation

– So far, the way we use `cudaMemcpy` serializes data transfer and GPU computation for `VecAddKernel()`



Device Overlap

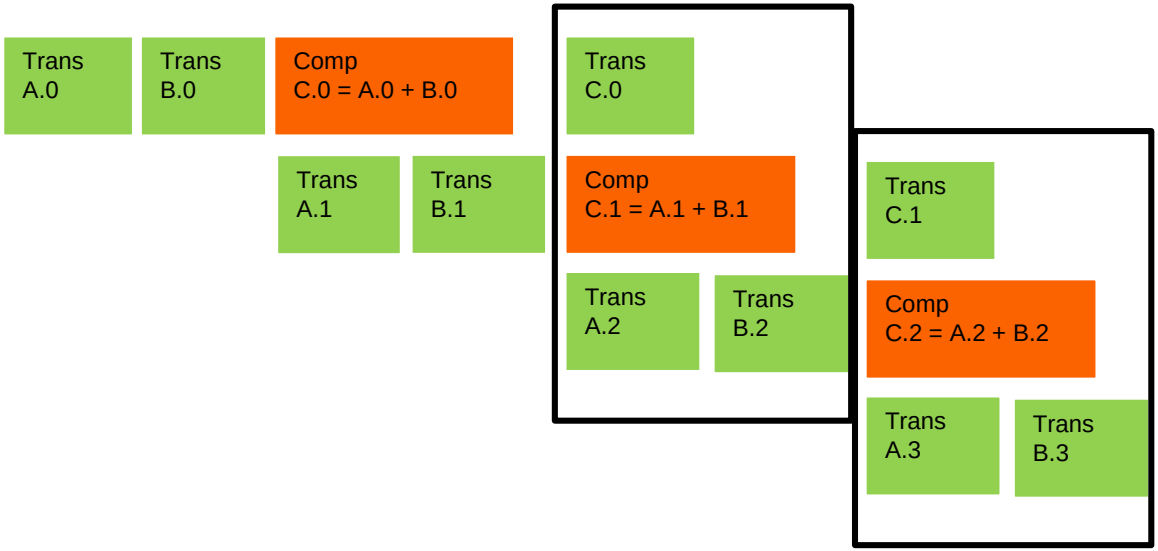
- Some CUDA devices support device overlap
  - Simultaneously execute a kernel while copying data between device and host memory

```
int dev_count;
cudaDeviceProp prop;

cudaGetDeviceCount( &dev_count);
for (int i = 0; i < dev_count; i++) {
  cudaGetDeviceProperties(&prop, i);
  if (prop.deviceOverlap) …
```

Ideal, Pipelined Timing

– Divide large vectors into segments
– Overlap transfer and compute of adjacent segments

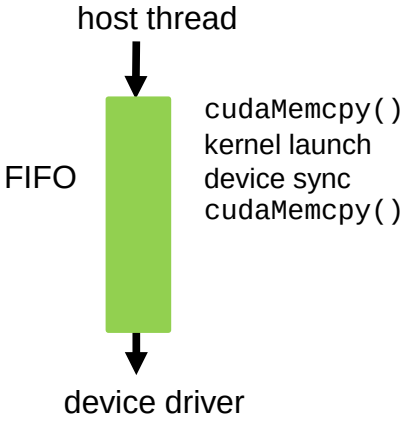


CUDA Streams

- CUDA supports parallel execution of kernels and `cudaMemcpy()` with “Streams”
- Each stream is a queue of operations (kernel launches and `cudaMemcpy()`calls)
- Operations (tasks) in different streams can go in parallel
  - “Task parallelism”

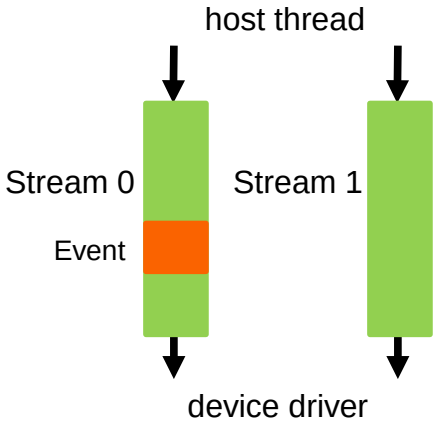
Streams

- Requests made from the host code are put into First-In-First-Out queues
  - Queues are read and processed asynchronously by the driver and device
  - Driver ensures that commands in a queue are processed in sequence. E.g., Memory copies end before kernel launch, etc.

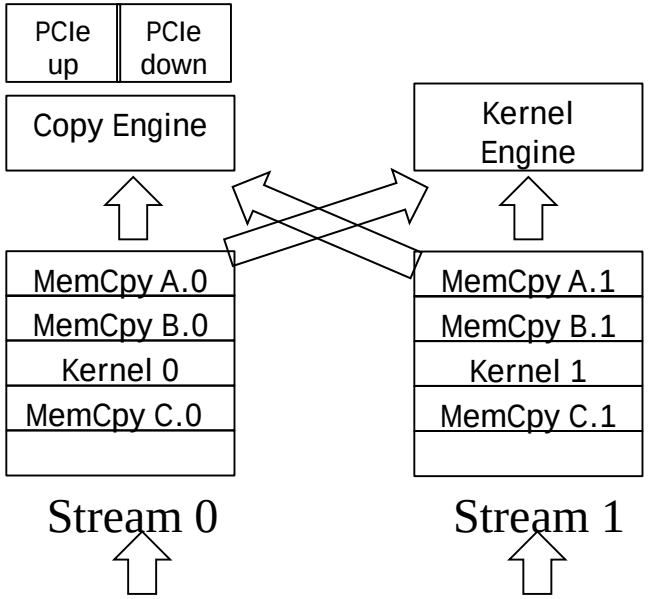


Streams cont.

– To allow concurrent copying and kernel execution, use multiple queues, called “streams”
  – CUDA “events” allow the host thread to query and synchronize with individual queues (i.e. streams).



Conceptual View of Streams



Operations (Kernel launches, `cudaMemcpy()` calls)

GPU Teaching Kit

Accelerated Computing