GPU Teaching Kit

Accelerated Computing

Module 14 – Efficient Host-Device Data Transfer

Lecture 14.3 - Overlapping Data Transfer with Computation

Objective

- To learn how to overlap data transfer with computation
  - Asynchronous data transfer in CUDA
  - Practical limitations of CUDA streams

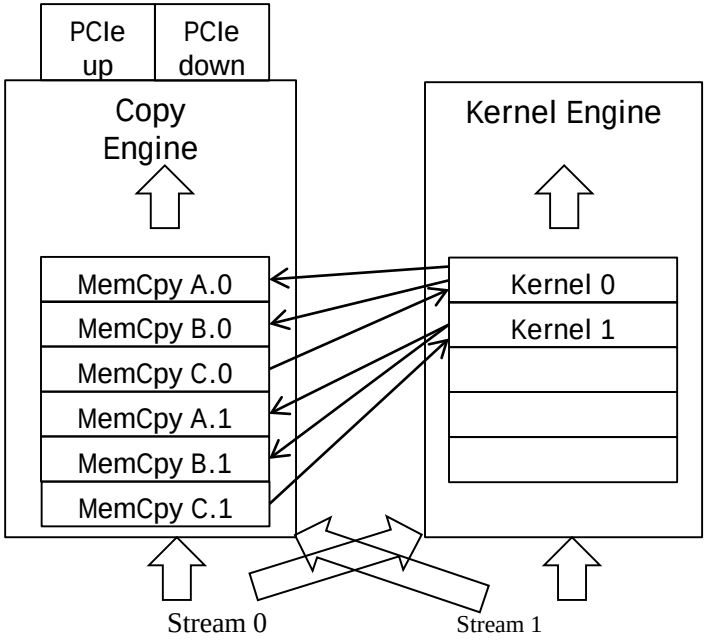
Simple Multi-Stream Host Code

```
cudaStream_t stream0, stream1;
cudaStreamCreate(&stream0);
cudaStreamCreate(&stream1);

float *d_A0, *d_B0, *d_C0; // device memory for stream 0
float *d_A1, *d_B1, *d_C1; // device memory for stream 1

// cudaMalloc() calls for d_A0, d_B0, d_C0, d_A1, d_B1, d_C1 go
here
```

Simple Multi-Stream Host Code (Cont.)

```
for (int i=0; i<n; i+=SegSize*2) {
    cudaMemcpyAsync(d_A0, h_A+i, SegSize*sizeof(float),…, stream0);
    cudaMemcpyAsync(d_B0, h_B+i, SegSize*sizeof(float),…, stream0);
    vecAdd<<<SegSize/256, 256, 0, stream0>>>(d_A0, d_B0,…);
    cudaMemcpyAsync(h_C+i, d_C0, SegSize*sizeof(float),…, stream0);
    cudaMemcpyAsync(d_A1, h_A+i+SegSize, SegSize*sizeof(float),…, stream1);
    cudaMemcpyAsync(d_B1, h_B+i+SegSize, SegSize*sizeof(float),…, stream1);
    vecAdd<<<SegSize/256, 256, 0, stream1>>>(d_A1, d_B1, …);
    cudaMemcpyAsync(d_C1, h_C+i+SegSize, SegSize*sizeof(float),…, stream1);
}
```

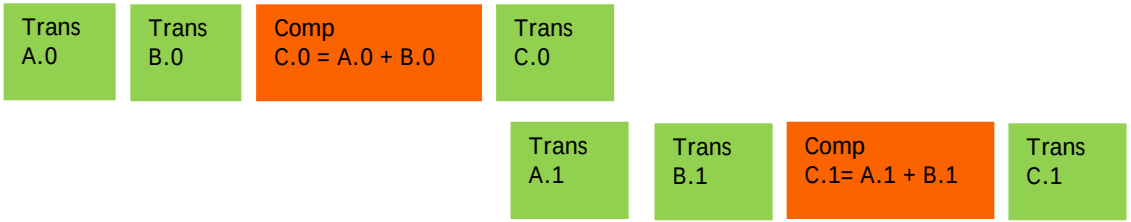
A View Closer to Reality in Previous GPUs



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

Not quite the overlap we want in some GPUs

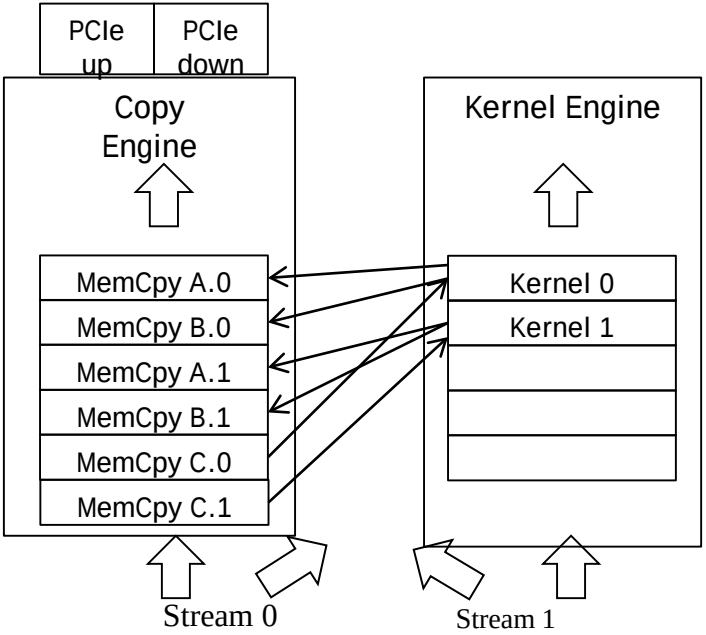
– C.0 blocks A.1 and B.1 in the copy engine queue



Better Multi-Stream Host Code

```
for (int i=0; i<n; i+=SegSize*2) {
  cudaMemcpyAsync(d_A0, h_A+i, SegSize*sizeof(float),…, stream0);
  cudaMemcpyAsync(d_B0, h_B+i, SegSize*sizeof(float),…, stream0);
  cudaMemcpyAsync(d_A1, h_A+i+SegSize, SegSize*sizeof(float),…, stream1);
  cudaMemcpyAsync(d_B1, h_B+i+SegSize, SegSize*sizeof(float),…, stream1);

  vecAdd<<<SegSize/256, 256, 0, stream0>>>(d_A0, d_B0, …);
  vecAdd<<<SegSize/256, 256, 0, stream1>>>(d_A1, d_B1, …);

  cudaMemcpyAsync(h_C+i, d_C0, SegSize*sizeof(float),…, stream0);
  cudaMemcpyAsync(h_C+i+SegSize, d_C1, SegSize*sizeof(float),…, stream1);
}
```

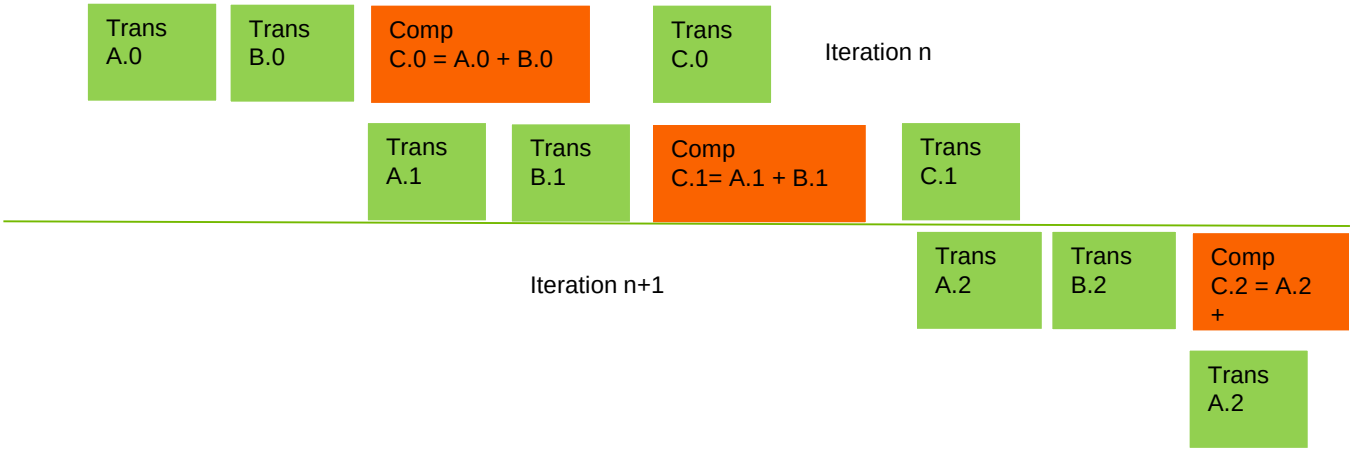
C.0 no longer blocks A.1 and B.1



Operations (Kernel launches, cudaMemcpy() calls)

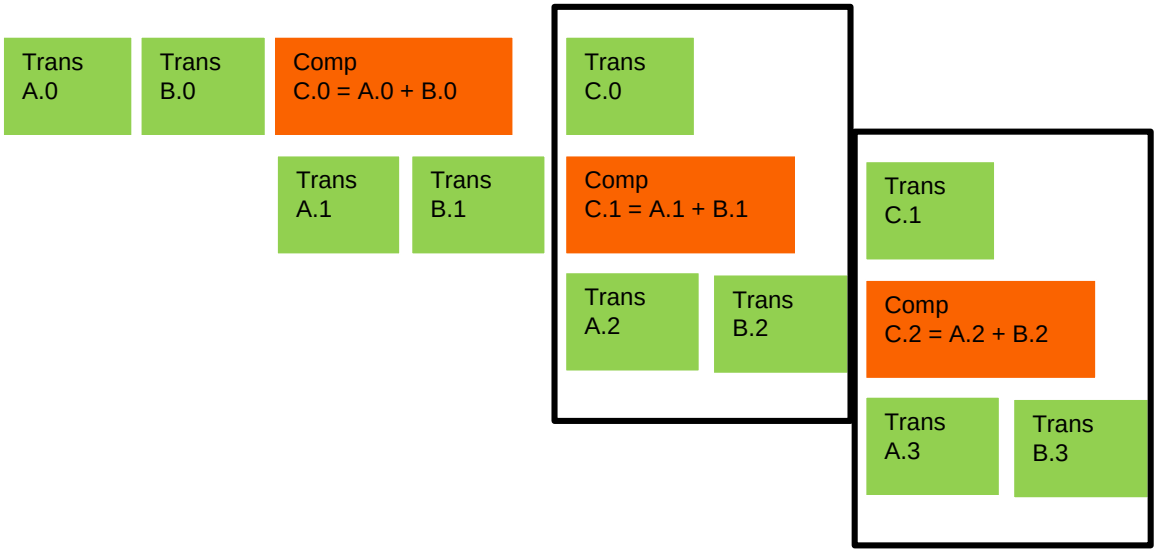
Better, not quite the best overlap

– C.1 blocks next iteration A.0 and B.0 in the copy engine queue



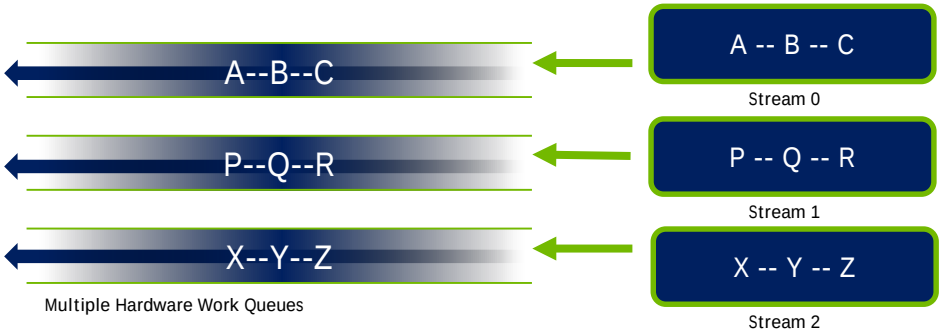
Ideal, Pipelined Timing

– Will need at least three buffers for each original A, B, and C, code is more complicated



Hyper Queues

– Provide multiple queues for each engine
– Allow more concurrency by allowing some streams to make progress for an engine while others are blocked



Wait until all tasks have completed

- `cudaStreamSynchronize(stream_id)`
  - Used in host code
  - Takes one parameter – stream identifier
  - Wait until all tasks in a stream have completed
  - `E.g., cudaStreamSynchronize(stream0)` in host code ensures that all tasks in the queues of stream0 have completed

- This is different from `cudaDeviceSynchronize()`
  - Also used in host code
  - No parameter
  - `cudaDeviceSynchronize()` waits until all tasks in all streams have completed for the current device

GPU Teaching Kit

Accelerated Computing