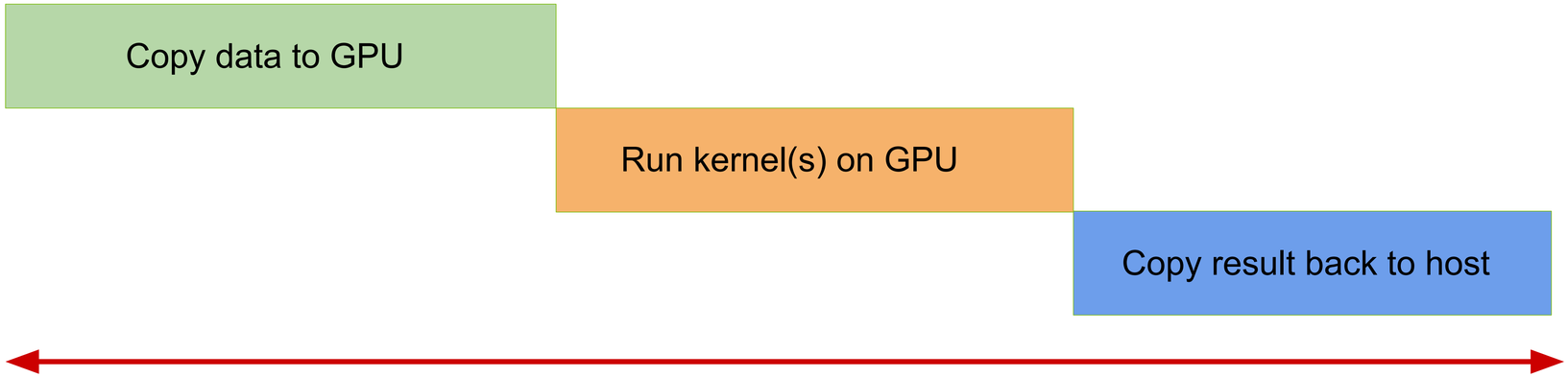
CUDA Streams

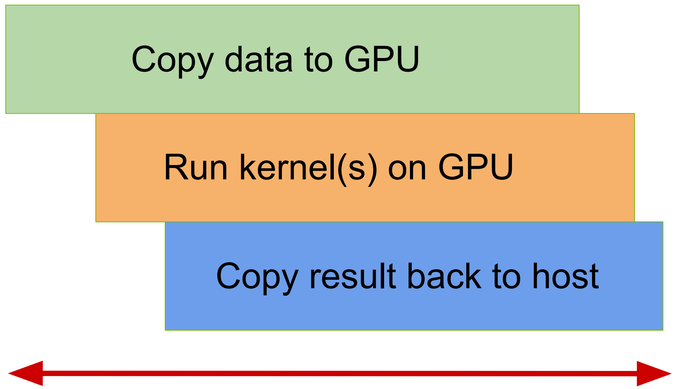
Matt Stack, Solutions Architect | 2/21/23

Concurrency- Motivation

- Normal CUDA implementation- 3 step process

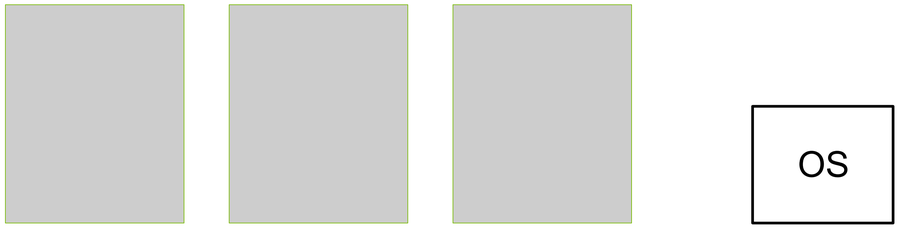


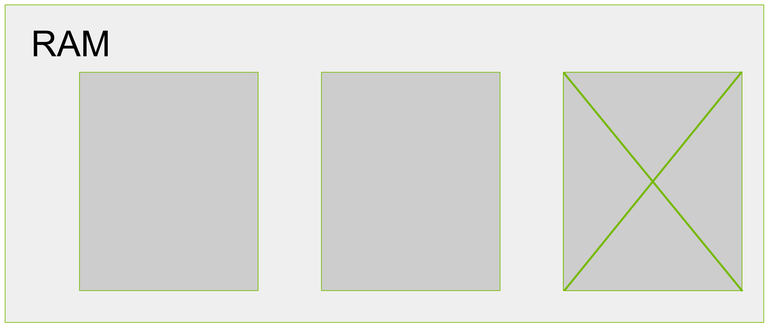
- Motivation: could we achieve a workflow like this?

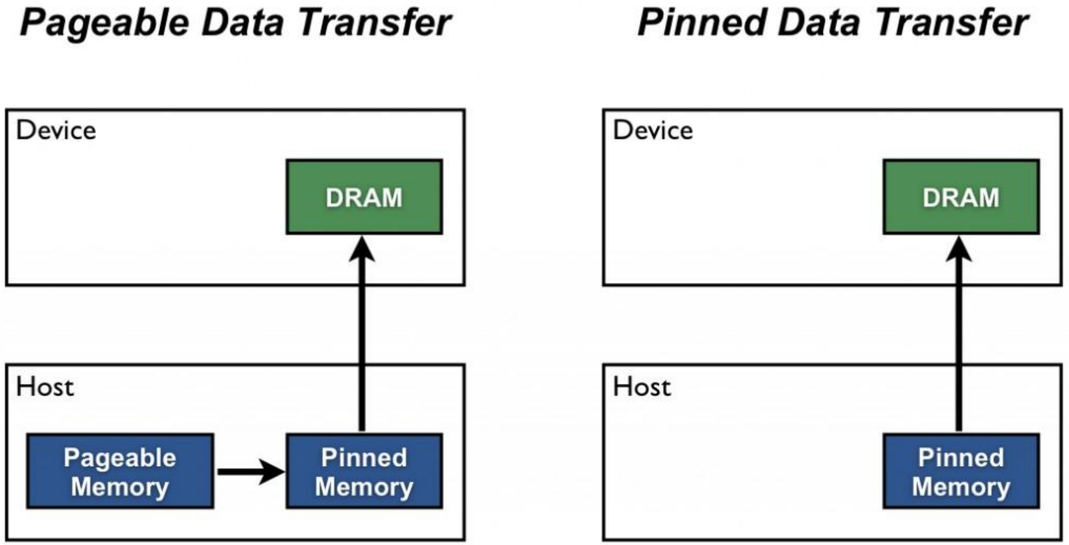


Pinned Memory

- Overview:
  - Faster Host to/from Device copies
  - Enables async memcpys to/from Host and Device
- API:
  - cudaMallocHost() and cudaHostAlloc()
  - cudaFreeHost()







Stream API Overview

- Default CUDA review:
  - Kernel are launched asynchronously with respect to Host
  - cudaMemcpy(HtoD/DtoH) blocks the Host thread, “blocking call”
  - Default stream

- Stream API:
  - cudaStreamCreate() / cudaStreamDestroy()
  - cudaMemcpyAsync()
    - (requires the Host addr to be pinned)
  - my_kernel<<<grid, block, 0, stream[i]>>>(…)

- Stream:
  - A Stream is a sequence of operations that are issued in-order

Stream Semantics

- Two operations* issued to the same stream *will execute in issue-order*. Operation B issued after Operation A will not begin until Operation A has completed

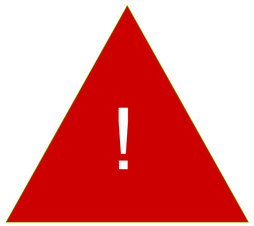
- Two operations* issued to different streams have *no order prescribed by CUDA.* Operation A issued in stream 1 may execute before, during, or after an Operation B issued in stream 2

*define Operation: usually cudaMemcpyAsync or a kernel<<< >>> launch, but there are other CUDA API calls that take a stream parameter

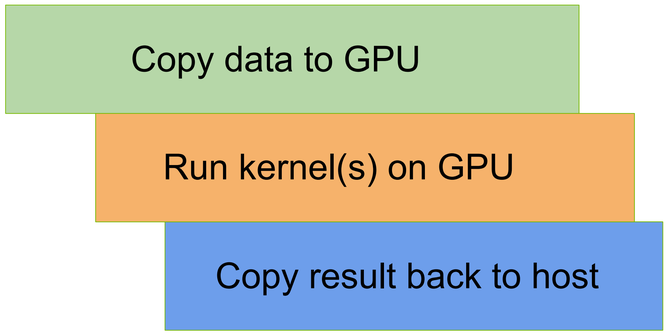
Example

```
cudaStream_t stream1, stream2;
cudaStreamCreate(&stream1);
cudaStreamCreate(&stream2);

cudaMemcpyAsync(dst, src, size, dir, stream1);
kernel<<< grid, block, 0, stream2 >>>(...);

cudaStreamSynchronize(stream2);
```

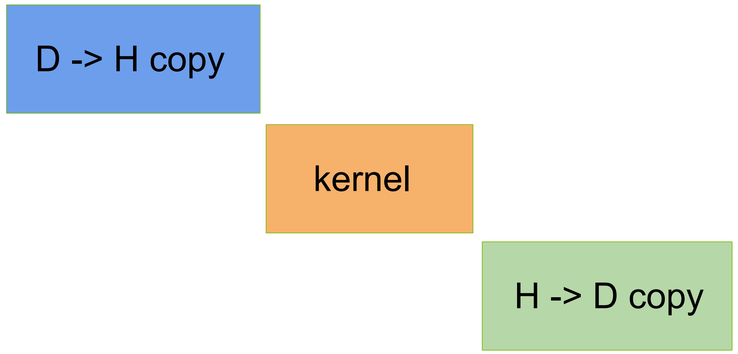
Potential overlap



Caution:
What happens in this example if the kernel depends on data from cudaMemAsync()?

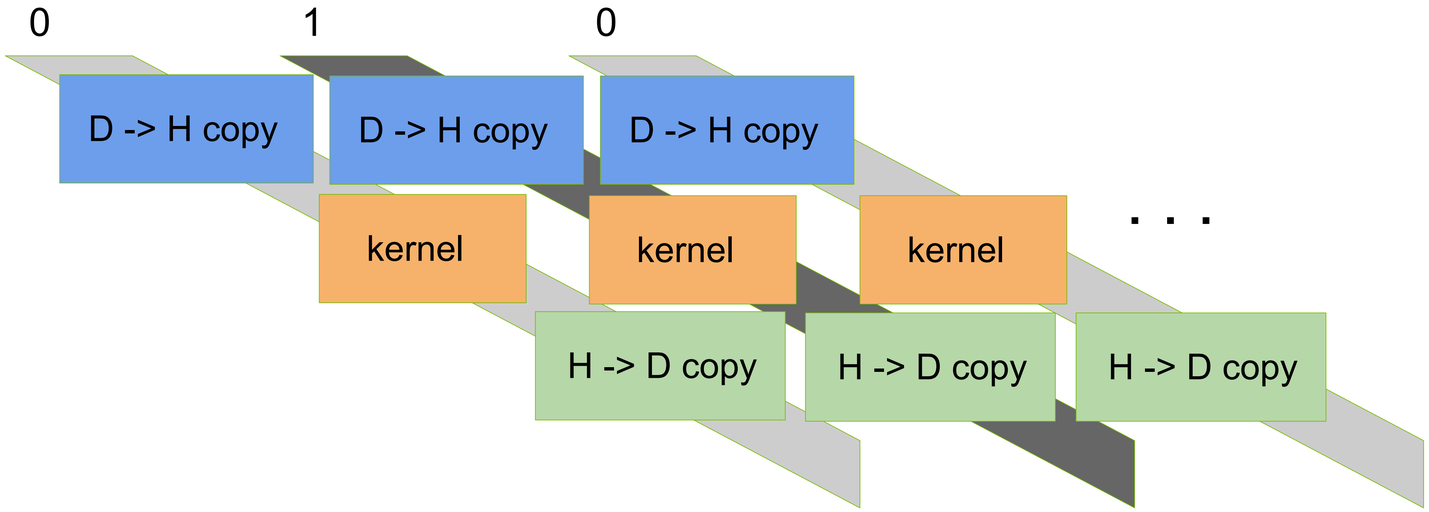


Example



Non-streams

```
cudaMemcpy(d_x, h_x, size_x,
cudaMemcpyHostToDevice);
kernel<<<blockspg, threadspb>>>(d_x, d_y, N);
cudaMemcpy(h_y, d_y, size_y,
cudaMemcpyDeviceToHost)
```

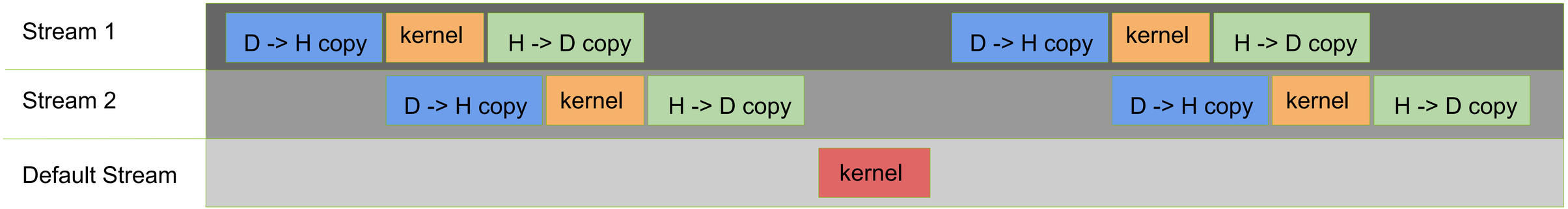


With streams

```
for (int i = 0; i < c; ++i){
   size_t offx = (size_x / c) * i;
   size_t offy = (size_y / c) * i;
   cudaMemcpyAsync(d_x+offx, h_x+offx, (size_x / c),
   cudaMemcpyHostToDevice, stream[i%ns]);
   Kernel<<< b/c, t, 0, stream[i%ns] >>>(d_x+offx,
   d_y+offy, N/c);
   cudaMemcpyAsync(h_y+offy, d_y+offy, (size_x / c),
   cudaMemcpyDeviceToHost, stream[i%ns]);
}
```

Default Stream

- Kernels<<<>>> and cudaMemcpy that don't specify a stream are sent to the Default Stream (Null stream)
- Operations submitted to the Default Stream will:
  - wait for all previously submitted work *in all streams* to finish before executing
  - not allow other operations in different streams to begin executing until it is finished



- Converting the Default Stream to an “ordinary” stream
  nvcc --default-stream per-thread …

CUDA Events

- CUDA Events are markers used for synchronization and signaling between streams and the Host
- Used for timing and complex synchronization between streams
- CudaEvents are “recorded” when they are issued
- CudaEvents are “completed” when the stream has reached the point where it was recorded

```
cudaEvent_t start, end;
cudaEventCreate(&start);
cudaEventCreate(&end);
cudaEventRecord(start); // “record” issued into default stream
Kernel<<<b, t >>>( … );
cudaEventRecord(stop);
cudaEventSynchronize(stop); // wait for stream activity to reach stop event
cudaEventElapsedTime(&float_time, start, stop);
```

- for stream signaling- cudaStreamWaitEvent(), makes a stream wait for an event to happen

Further Reading

- Going into more technical detail with queues and depthvsbreadth first:
  - https://developer.download.nvidia.com/CUDA/training/StreamsAndConcurrencyWebinar.pdf

- APIs for Streams and Events
  - https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__STREAM.html
  - https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__EVENT.html#group__CUDART__EVENT

- CUDA Programming Guide is always a good place to get a overview of a feature:
  - https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#streams

- Cuda Streams performance best practices:
  - https://on-demand.gputechconf.com/gtc/2014/presentations/S4158-cuda-streams-best-practices-common-pitfalls.pdf

- CUDA Streams lecture
  - https://www.olcf.ornl.gov/calendar/cuda-concurrency/ (CUDA Concurrency on Vimeo)

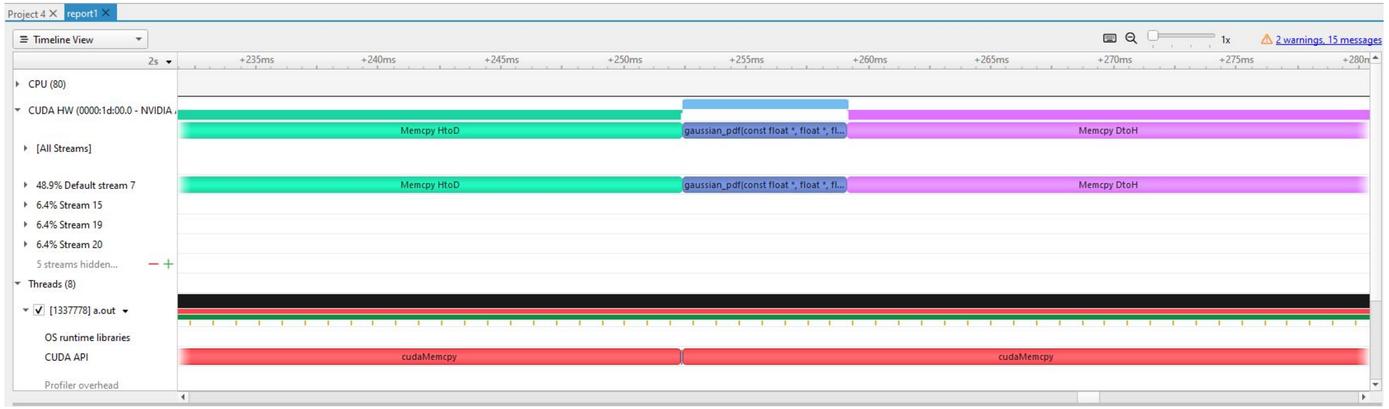
Hands on

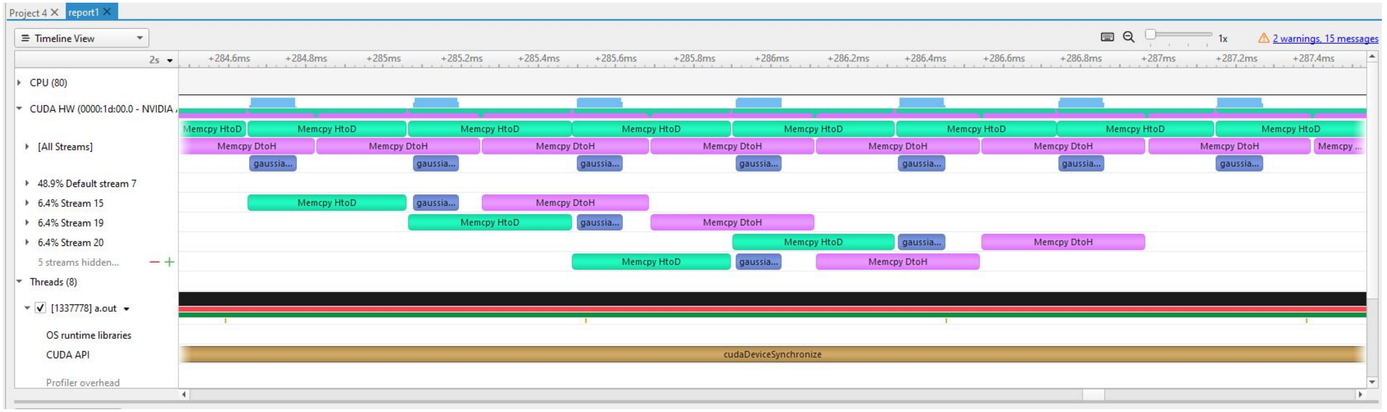
https://github.com/matt-stack/TAC-HEP-Training-Feb2023.git

Ex1. Compute/Copy overlap

Ex2. Cuda Event Timing

Ex3. Default streams with Nsys






NVIDIA