Optimizing Host-Device Data Communication II - *CUDA Streams & Asynchronous Copy*

Stefano Markidis

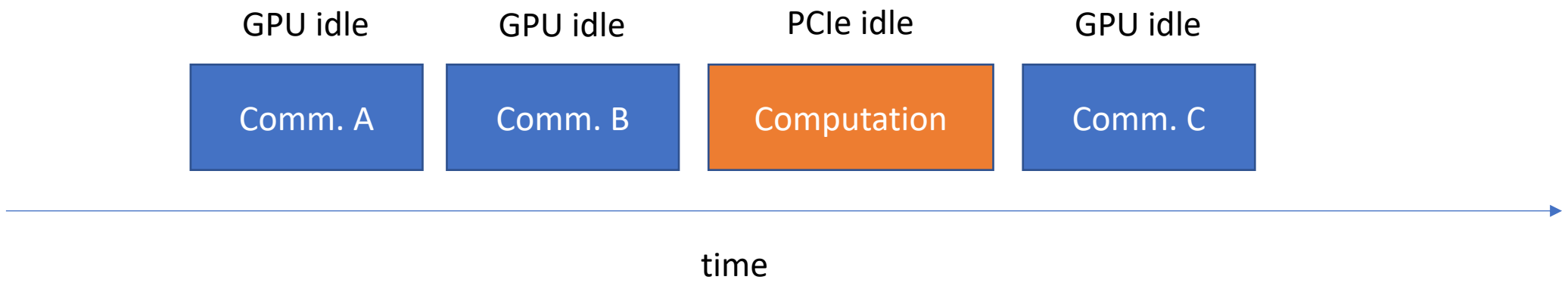
Three Key-Points

1. CUDA supports parallel execution of kernel and memcpy by using Streams that are a queue of operations
2. We create/destroy streams with `cudaStreamCreate()`/`cudaStreamDestroy()` and we use `cudaMemcpyAsync()` to copy asynchronously on a stream
3. When launching a kernel on a stream we use the stream identifier as fourth parameter of the execution configuration

Serialized Data Communication and Computation

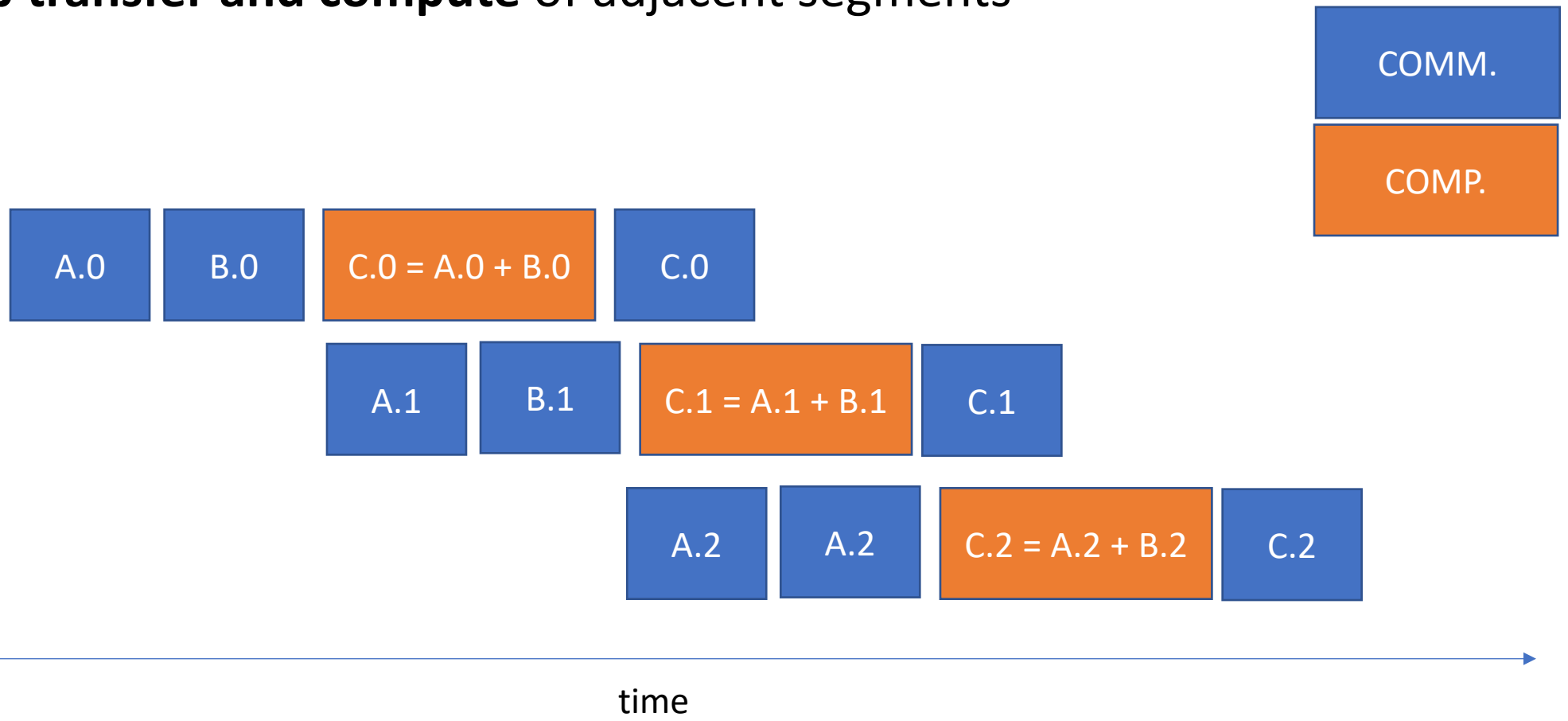
- So far, the way we use `cudaMemcpy` serializes data transfer and GPU computation ()

```
...
cudaMemcpy(d_A, A, numBytes, cudaMemcpyHostToDevice);
cudaMemcpy(d_B, B, numBytes, cudaMemcpyHostToDevice);
kernel_add<<<1,N>>>(d_A,d_B,...);
cudaMemcpy(C, d_C, numBytes, cudaMemcpyDeviceToHost);
...
```



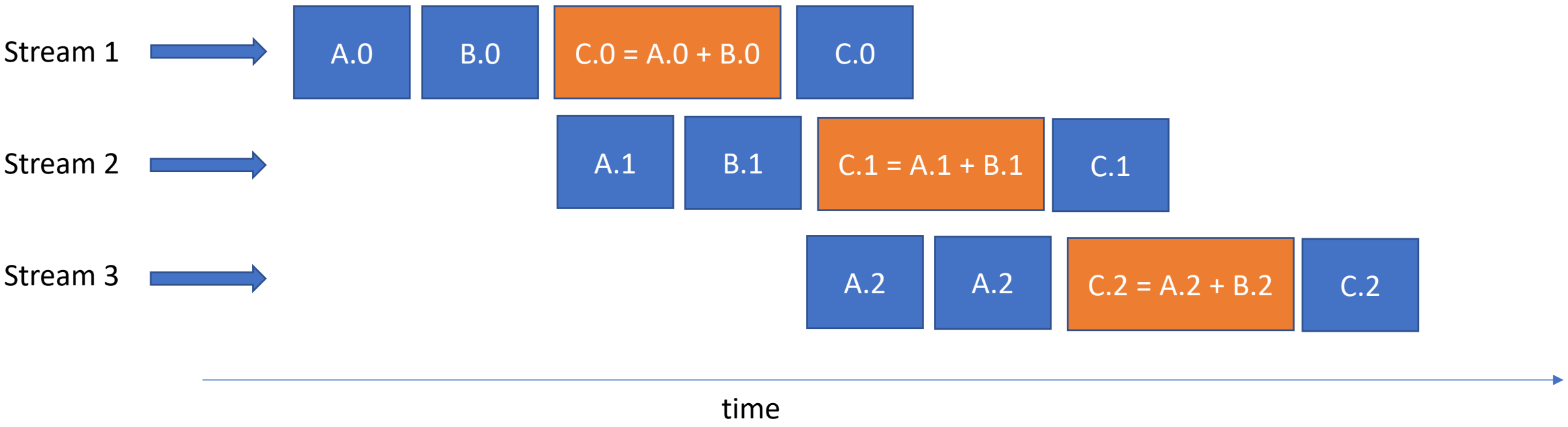
Can we do better? Pipeline Communication & Compute

- **Divide vectors into segments**
  - Example divide in three segments: A = A.0 A.1 A.2, B = B.0 B.1 B.2, C = C.0 C.1 C.3
- **Overlap transfer and compute** of adjacent segments



How do we do that? Cuda Streams

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



The Default Stream

- All device operations (kernels and data transfers) in CUDA run in a stream.
  - When no stream is specified, the **default stream** (also called the *null* stream) is used.
- The default stream is different from other streams.
- It is a **synchronizing stream with respect to operations on the device**
  - no operation in the default stream will begin until all previously issued operations *in any stream on the device* have completed
  - an operation in the default stream must complete before any other operation (in any stream on the device) will begin.

Default Stream

```
...
cudaMemcpy(d_A, A, numBytes, cudaMemcpyHostToDevice);
cudaMemcpy(d_B, B, numBytes, cudaMemcpyHostToDevice);
kernel_add<<<1,N>>>(d_A,d_B,...);
cudaMemcpy(C, d_C, numBytes, cudaMemcpyDeviceToHost);
...
```

Non-Default CUDA Streams

Non-default streams in CUDA C/C++ are declared, created, and destroyed in host code like this:

```
...
cudaStream_t stream1;
cudaStreamCreate(&stream1);

...

cudaStreamDestroy(stream1);
...
```

Data Transfer to a Non-Default Stream

```
cudaMemcpyAsync(d_a, a, N, cudaMemcpyHostToDevice, stream1)
```

- To issue a data transfer to a non-default stream we use the `cudaMemcpyAsync()`
  - similar to the `cudaMemcpy()` but takes a **stream identifier as a fifth argument**
- `cudaMemcpyAsync()` is **non-blocking on the host**, so control returns to the host thread immediately after the transfer is issued.

Kernel on a Non-Default Stream

```
kernel_add<<<1,N,0,stream1>>>(d_A,d_B,...);
```

- To issue a kernel to a non-default stream we specify the stream identifier as a fourth execution configuration parameter
  - The third execution configuration parameter allocates shared device memory, which is `0` in this example

Overlapping Kernel Execution and Data Transfers

Two requirements:

- The kernel execution and the data transfer to be overlapped must both occur in **different non-default streams**.
- The host memory involved in the data transfer must be **pinned memory**.

Synchronization of Streams

```
cudaStreamSynchronize(stream_id)
```

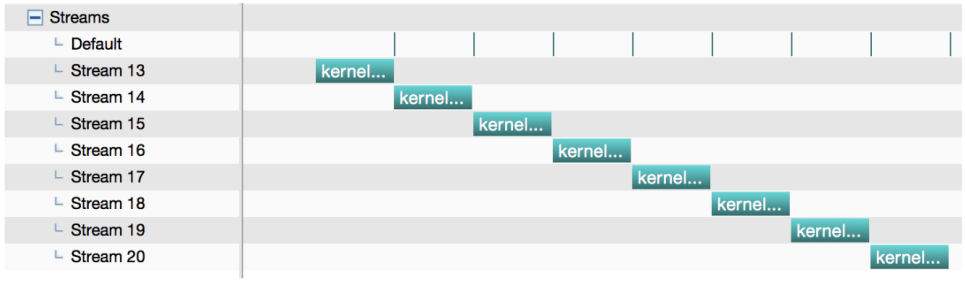
- Used in host code
- Takes one parameter: the stream identifier
- Wait until **all tasks in a stream** have completed
- This is different from `cudaDeviceSynchronize()`
  - Also used in host code
  - No parameter
  - `cudaDeviceSynchronize()` waits until all tasks in all streams have completed for the current device

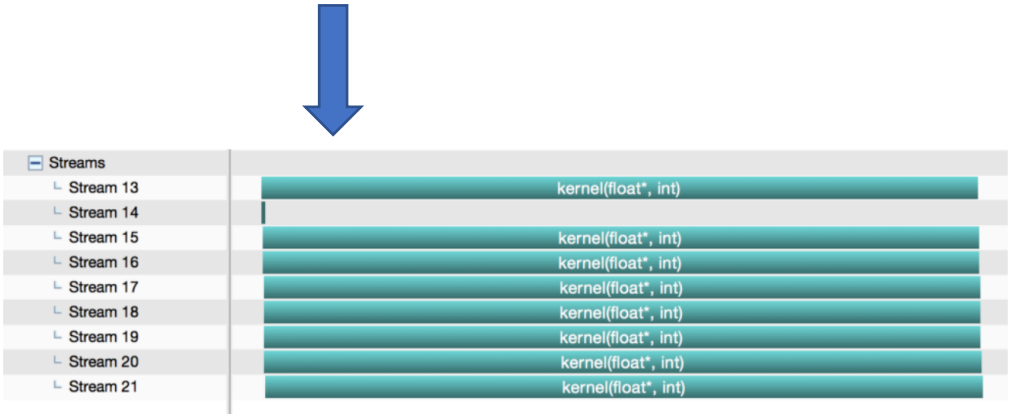
CUDA7 Streams – Per-Thread Default Stream

- Before CUDA 7, each device has a single default stream used for all host threads, which causes implicit synchronization of all streams.
- CUDA 7 introduces a new option, the *per-thread default stream*
  - Each host thread its own default stream.

```
nvcc --default-stream per-thread ./stream_test.cu -o stream_per-thread
```

Before per-thread default stream





To Summarize

- CUDA supports parallel execution of kernel and memcpy operations with Streams that are queue operations
- We create/destroy streams with `cudaStreamCreate()`/`cudaStreamDestroy()` and we use `cudaMemcpyAsync()` to copy asynchronously on a stream
- When launching a kernel on a stream we use the stream identifier as fourth parameter of the execution configuration