CME213/ME339
Lecture 10

Eric Darve
Erich Elsen
Austin Gibbons

Department of Mechanical Engineering
Institute for Computational and Mathematical Engineering
Stanford University

Spring 2012

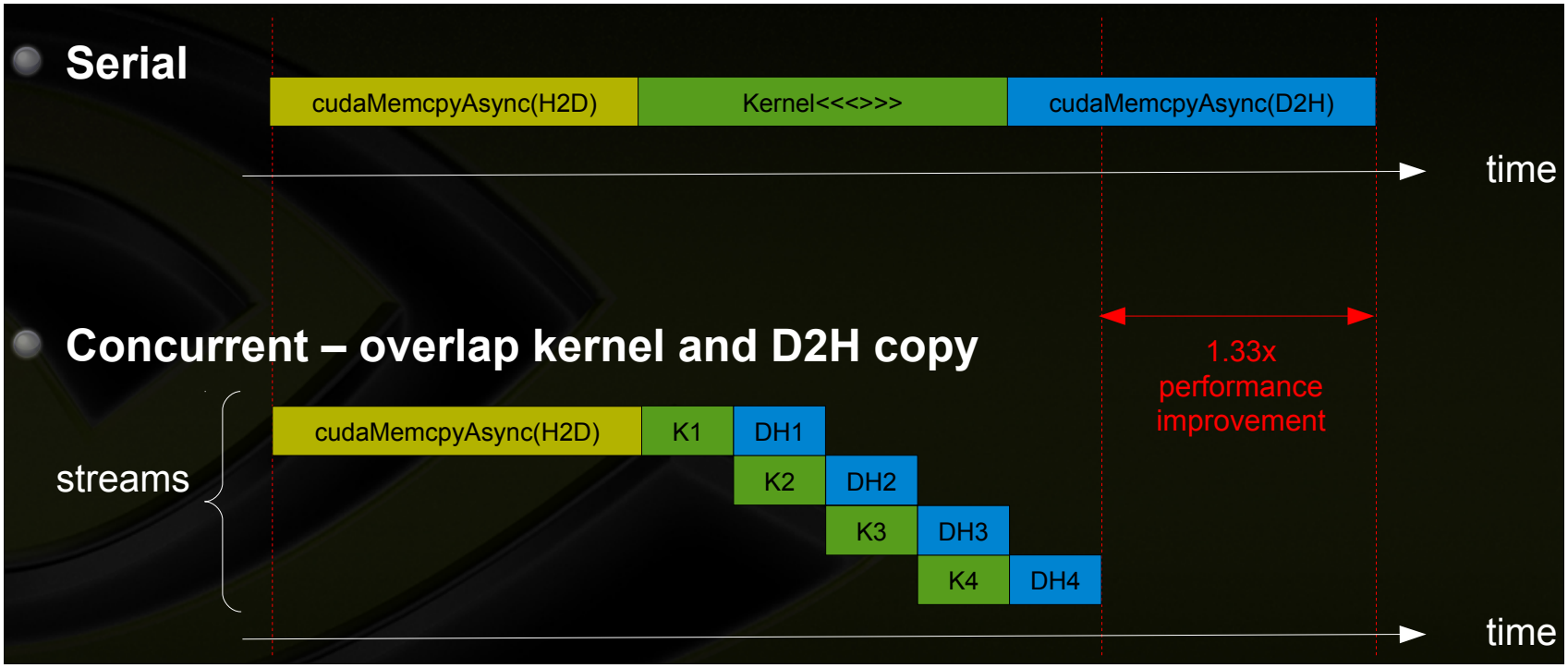
Streams

- A relatively recent addition.
- An advanced topic.
- What it does: it allows to run concurrently
  1. host/device memory copies (`Memcpy`)
  2. CUDA kernels
  3. CPU
- Google "`gpu computing webinar`" and look for "CUDA Concurrency & Streams" (PDF and video)

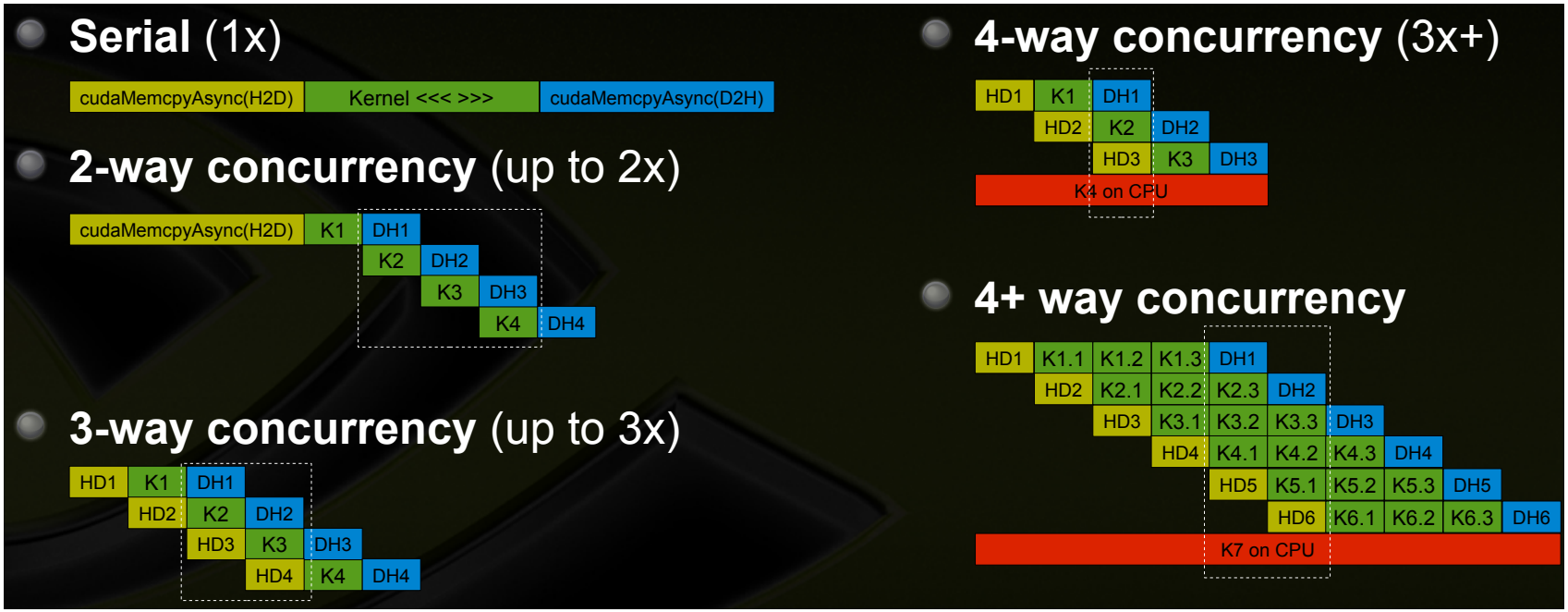
What is a stream?

- Stream: sequence of CUDA operations that are issued by the CPU and that will be executed sequentially on the device.
- CUDA operations in different streams may run concurrently, or be interleaved.

Basic Example

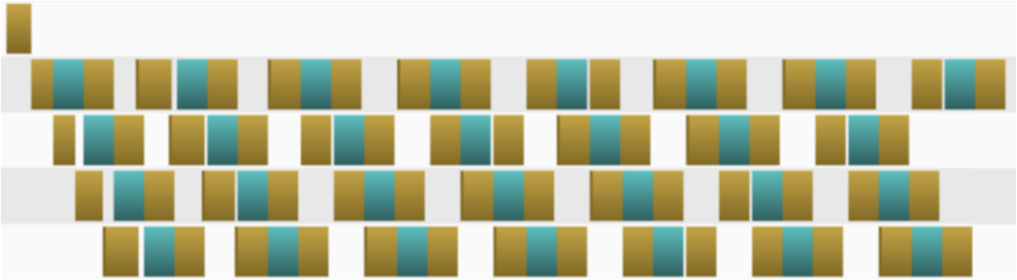


HD, DH, K and CPU overlap



DGEMM

- DGEMM: m=n=8192, k=288.
- GPU is fast but one needs to move the data to GPU, compute and move back. `Memcpy` are typically slow. Remedy: communicate a bit, start computing immediately, communicate the rest while the calculation is on-going.
- CPU (4 core Westmere x5670 @2.93 GHz, MKL): 43 Gflops
- GPU (C2070)
  - Serial: 125 Gflops (2.9x)
  - 2-way: 177 Gflops (4.1x)
  - 3-way: 262 Gflops (6.1x)
- GPU + CPU, 4-way: 282 Gflops (6.6x)
- All communication hidden; effectively removes device memory size limitation.



Default stream

- For compatibility issues, there is a default stream (0) that does not behave like the others.
- The default stream is the stream used when no stream is specified (what we have been doing up to now without knowing it).
- Synchronous with respect to device.
- Synchronous with respect to host except for:
  - Kernel launches
  - `cudaMemcpyAsync`
  - `cudaMemsetAsync`

Requirements for concurrency

- CUDA operations must be in different, non-0, streams.
- `cudaMemcpyAsync` with host from 'pinned' memory:
  - Page-locked memory
  - Allocated using `cudaMallocHost()` or `cudaHostAlloc()`
- `cudaMemcpyAsyncs` in different directions

Code example

```
cudaStream_t stream1, stream2, stream3, stream4;
cudaStreamCreate(&stream1);
...
cudaMalloc(&dev1, size);
cudaMallocHost(&host1, size);
...
cudaMemcpyAsync(dev1, host1, size, H2D, stream1);
kernel2<<<grid, block, 0, stream2>>>( ..., dev2, ... );
kernel3<<<grid, block, 0, stream3>>>( ..., dev3, ... );
cudaMemcpyAsync(host4, dev4, size, D2H, stream4);
some_CPU_method();
```

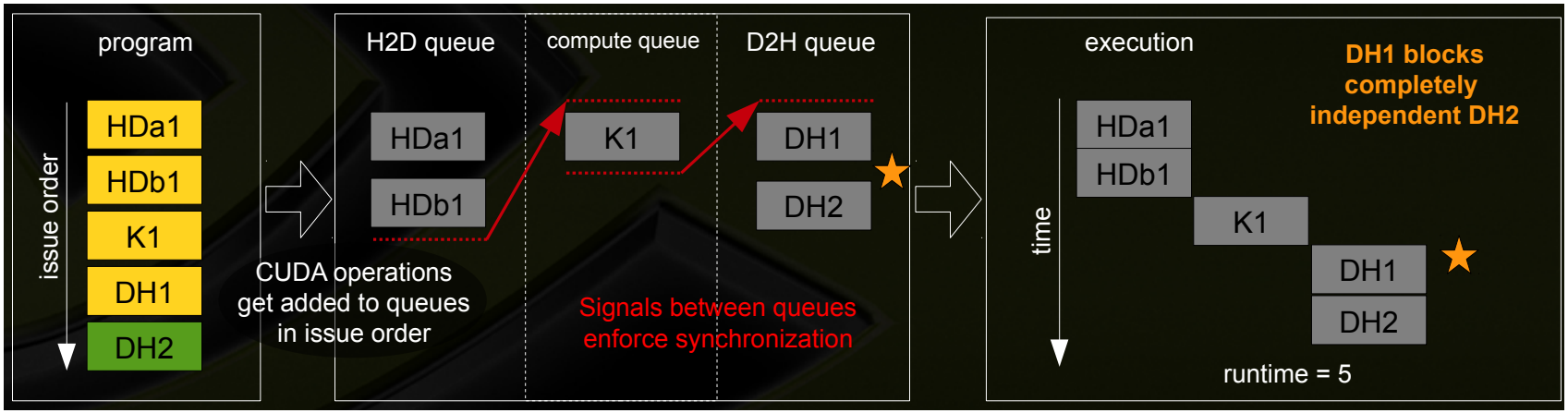
Synchronization

- Because of the possible concurrent execution, we now need some ways to synchronize at certain points when necessary.
- Synchronize everything: `cudaDeviceSynchronize()`; blocks host until all issued CUDA calls are complete.
- Synchronize w.r.t. a specific stream: `cudaStreamSynchronize(streamid)`; blocks host until all CUDA calls in `streamid` are complete.
- Synchronize using events:
  - `cudaEventRecord(event, streamid)`: event is recorded after all preceding operations in stream have been completed.
  - `cudaEventSynchronize(event)`: host waits for event.
  - `cudaStreamWaitEvent(stream, event)`: stream waits for event.
  - `cudaEventQuery(event)`: query the state of an event.

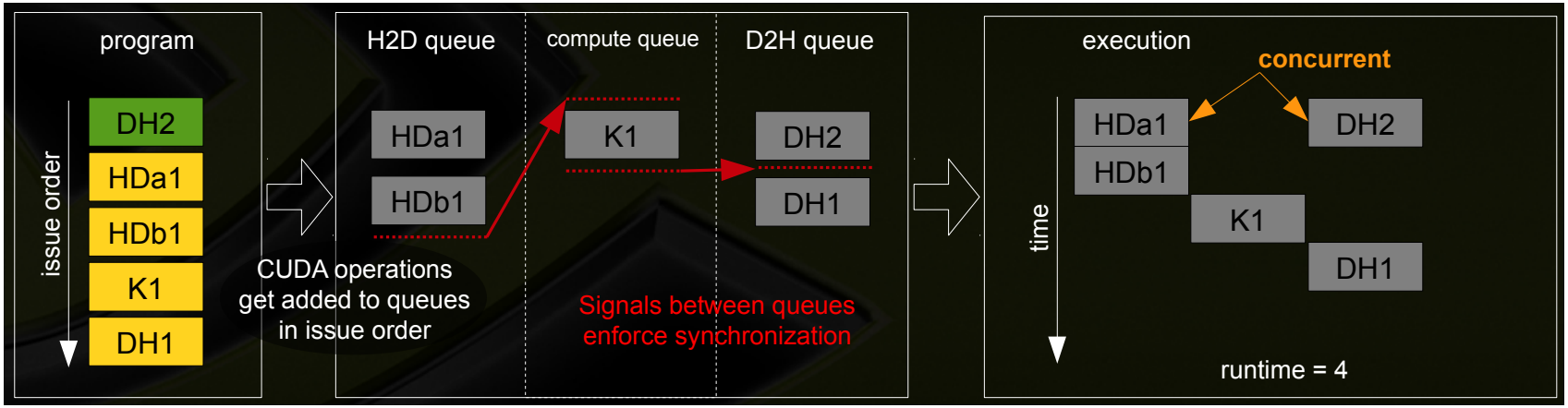
Understanding stream scheduling

- Fermi hardware has 3 queues:
  1. Compute engine queue
  2. H2D copy engine queue
  3. D2H copy engine queue
- CUDA operations are dispatched to HW in the sequence they were issued, and placed in the relevant queue.
  - Stream dependencies between engine queues are maintained, but lost within an engine queue.
- A CUDA operation is dispatched from the engine queue if:
  - Preceding calls in the same stream have completed.
  - Preceding calls in the same queue have been dispatched.
  - Resources are available.
- A blocked operation blocks all other operations in the queue, even in other streams.

Blocked queue because of queue dependency

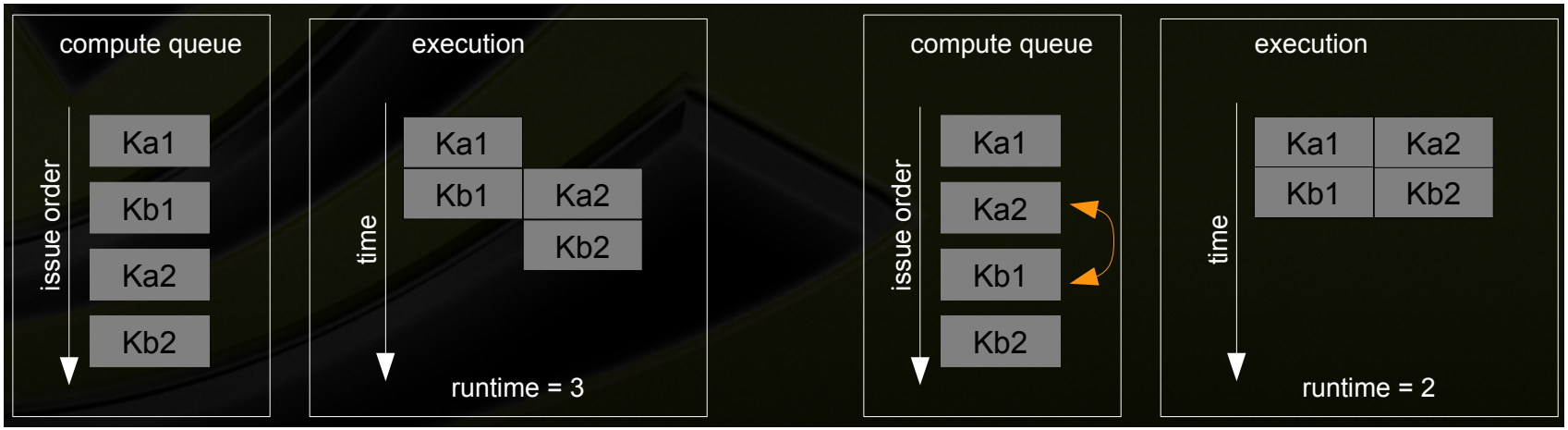


Blocked queue — fixed



Issue order of kernels

Issue order matters: depth-first or breadth first.



More information

More details available on-line.

Google "`gpu computing webinar`" and look for "CUDA Concurrency & Streams" (PDF and video) by Steve Rennich.

Section 3.2.5 in Programming Guide and Chapter 6 in Best Practices Guide.