GPU PROGRAMMING WITH CUDA

Streams and Events

25. – 29. April 2022 | Kaveh Haghighi Mood, Jochen Kreutz | JSC

JÜLICH
Forschungszentrum

OVERVIEW

- Manual memory management
- Pinned (pagelocked) host memory
- Asynchronous and concurrent memory copies
- Cuda Streams
  - Default stream and `cudaStreamNonBlocking` flag
- Cuda Events

GETTING DATA IN AND OUT

GPU has separate memory, common workflow for manual data transfers:

- Allocate memory on device
- Transfer data from host to device
- Run your kernels
- Transfer data from device to host
- Free device memory

GETTING DATA IN AND OUT

allocate device memory

```
cudaMalloc ( void** pointer, size_t nbytes )
```

Example

```
// Allocate a vector of 2048 floats on device
float* a_gpu;
int n = 2048;
cudaMalloc ( &a_gpu, n * sizeof ( float ) );
```

GETTING DATA IN AND OUT

Copy from host to device

copy data from host to device memory

```
cudaMemcpy ( void* dst, void* src, size_t nbytes,
             enum cudaMemcpyKind direction )
```

Example

```
// Copy vector a of 2048 floats to a_gpu on device
cudaMemcpy ( a_gpu, a, 2048 * sizeof ( float ), cudaMemcpyHostToDevice );
```

GETTING DATA IN AND OUT

Copy from device to host

copy data from device to host memory

```
cudaMemcpy ( void* dst, void* src, size_t nbytes,
             enum cudaMemcpyKind direction )
```

Example

```
// Copy device vector a_gpu of 2048 floats to a  on host
cudaMemcpy ( a, a_gpu, 2048 * sizeof ( float ), cudaMemcpyDeviceToHost );
```

GETTING DATA IN AND OUT

Manual data management

Allocate memory on device

```
cudaMalloc ( void** pointer, size_t nbytes )
```

Transfer data between host and device

```
cudaMemcpy ( void* dst, void* src, size_t nbytes,
             enum cudaMemcpyKind direction )
```

Free device memory

```
cudaFree ( void* pointer )
```

PINNED HOST MEMORY

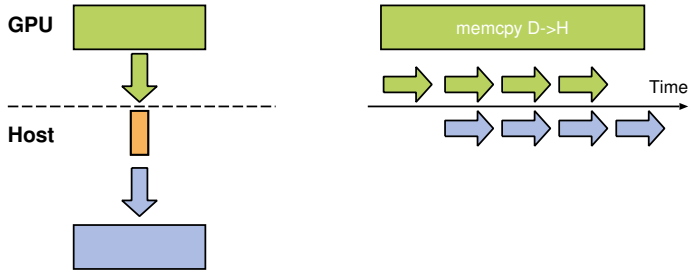
host memory allocated with malloc is pagable

- memory pages associated with the memory can be moved around by the OS kernel, e.g. to swap space on hard disk

transfers to and from the GPU memory need to go over PCIe

- PCIe transfers are handled by DMA engines on the GPU and work independently of the CPU / OS kernel
- if OS kernel moves pages involved in such a DMA transfer the wrong data will be moved
- pinning memory pages inhibits the OS kernel from moving them around and makes them usable for DMA transfers

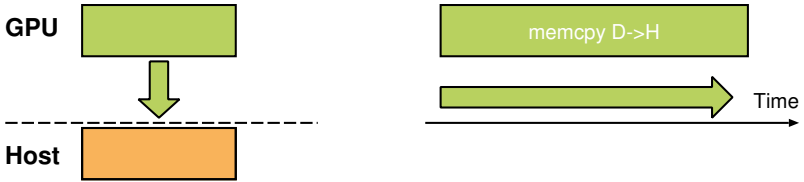
PINNED HOST MEMORY



Host memory allocated with malloc is staged through a pinned memory buffer managed by the Cuda driver

- No asynchronous memory copies are possible (CPU interaction is necessary to drive the pipeline)
- Higher latency and lower bandwidth compared to DMA transfers

PINNED HOST MEMORY



Using pinned host memory

- Enables asynchronous memory copies
- Lowers latency and increases bandwidth

PINNED HOST MEMORY

How to use

- Using POSIX functions like mlock is not sufficient, because the Cuda driver needs to know that the memory is pinned
- Two ways to get pinned host memory
  - Using `cudaMallocHost / cudaFreeHost` to allocate pinned memory
  - Using `cudaHostRegister / cudaHostUnregister` to pin memory after allocation

PINNED HOST MEMORY

How to use

- Using POSIX functions like mlock is not sufficient, because the Cuda driver needs to know that the memory is pinned
- Two ways to get pinned host memory
  - Using `cudaMallocHost / cudaFreeHost` to allocate pinned memory
  - Using `cudaHostRegister / cudaHostUnregister` to pin memory after allocation
- `cudaMemcpy` makes automatic use of it
- `cudaMemcpyAsync` can be used to issue asynchronous memory copies

CUDA STREAMS

How to use

- Cuda Streams are work queues to express concurrency between different tasks, e.g.
  - Host to device memory copies
  - Device to host memory copies
  - Kernel execution

CUDA STREAMS

How to use

- Cuda Streams are work queues to express concurrency between different tasks, e.g.
  - Host to device memory copies
  - Device to host memory copies
  - Kernel execution

- To overlap different tasks just launch them in different streams
  - All tasks launched into the same stream are executed in order
  - Tasks launched into different streams might execute concurrently (depending on available resoucres: copy engines, compute resources)

```
Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
Maximum memory pitch:                         2147483647 bytes
Texture alignment:                            512 bytes
Concurrent copy and kernel execution:         Yes with 7 copy engine(s)
Run time limit on kernels:                    No
Integrated GPU sharing Host Memory:           No
```

CUDA STREAMS

How to use

- Create / destroy a stream

```
cudaStream_t stream;
cudaStreamCreate ( &stream );
cudaStreamDestroy ( stream );
```

- Launch

```
my_kernel <<< grid, block, 0, stream >>> ( ... );
cudaMemcpyAsync (... , stream );
```

- Synchronize

```
cudaStreamSynchronize ( stream );
```

CUDA STREAMS

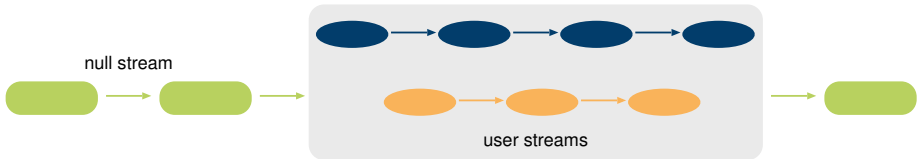
The default (NULL) stream

Kernel launches are always asynchronous

- Which stream is used here ?

```
my_kernel <<< grid, block >>> ( ... ),
```

- The default (NULL) stream is used

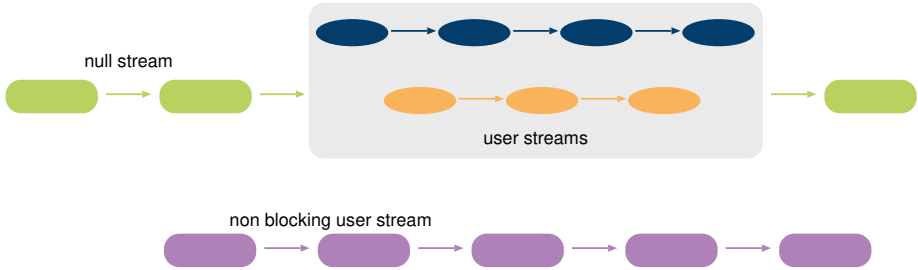


JÜLICH Forschungszentrum

CUDA STREAMS

The default (NULL) stream

- The default (NULL) stream waits for work in all other streams which do not have the cudaStreamNonBlocking flag set



- User streams with the cudaStreamNonBlocking flag set can execute concurrently to the default stream

CUDA EVENTS

Cuda Events are synchronization markers that can be used to:

- Time asynchronous tasks in streams
- Allow fine grained synchronization within a stream
- Allow inter stream synchronization, e.g. let a stream wait for an event in another stream

CUDA EVENTS

How to use

- Create / destroy events

```
cudaEvent_t event;
cudaEventCreate ( &event );
cudaEventDestroy ( event );
```

- Record

```
cudaEventRecord ( event, stream );
```

- Query

```
cudaEventQuery ( event );
```

- Synchronize

```
cudaEventSynchronize ( event );
```

CUDA EVENTS

How to use for kernel timing

Example

```
cudaEventRecord ( startEvent, stream );
my_kernel <<< grid, block, 0, stream >>> ( ... );
cudaEventRecord ( endEvent, stream );

//Host can do other work in between

float runtime = 0.0f;
cudaEventSynchronize ( endEvent );
cudaEventElapsedTime ( &runtime, startEvent, endEvent );
```

USING CUBLAS WITH STREAMS

Example

```
#include "cublas_v2.h"
...
cublasHandle_t handle;

//Initialize cuBLAS
cublasCreate ( &handle );
//Set cuBLAS execution stream
cublasSetStream ( handle, stream );
//Call cuBLAS routine, e.g.: SAXPY
cublasSaxpy ( handle, n, &alpha, x, 1, y, 1);
...
// Free resources
cublasDestroy ( handle );
```

EXERCISE

Asynchronous data transfers using streams

Instructions:

.../exercises/tasks/Instructions.ipynb

- *task1a:* using pinned memory
- *task1b:* asynchronous memcpy using streams
- *task2:* CUBLAS example with timing