CUDA STREAMS AND EVENTS

25. APRIL 2018 | JOCHEN KREUTZ

OVERVIEW

- Manual memory management
- Pinned (pagelocked) host memory
- Asynchronous and concurrent memory copies
- CUDA streams
  - The default stream and the `cudaStreamNonBlocking` flag
- CUDA Events
- CUBLAS
- nvprof + nvvp recap

GETTING DATA IN AND OUT

- GPU has separate memory
- Allocate memory on device
- Transfer data from host to device
- Transfer data from device to host
- Free device memory

GETTING DATA IN AND OUT

Allocate device memory

```
cudaMalloc ( T** 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) );
```

Address of pointer

Get size of a float

JÜLICH Forschungszentrum

COPY FROM HOST TO DEVICE

Copy data form host to device memory
`cudaMemcpy(void* dst, void* src, size_t nbytes, enum cudaMemcpyKind dir)`

Example:

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

COPY FROM DEVICE TO HOST

Copy data form host to device memory

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

Example:

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

GETTING DATA IN AND OUT

- 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  dir)
//dir is cudaMemcpyHostToDevice or cudaMemcpyDeviceToHost
```

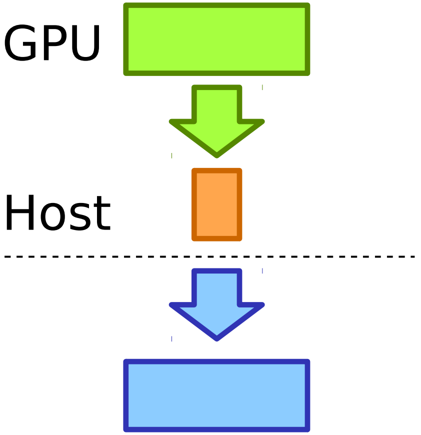
- Free device memory

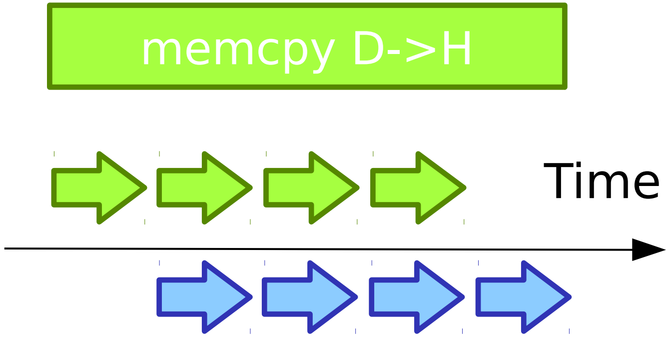
```
cudaFree(void* pointer)
```

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 PCI-E
    - PCI-E transfers are handled by DMA engines on the GPU and work independently of the CPU/OS kernel
        - If OS kernel moves memory pages involved in such a DMA transfer the wrong data will be moved
        - Pinning memory pages inhibit the OS kernel from moving them around and make them usable to DMA transfer

PINNED HOST MEMORY

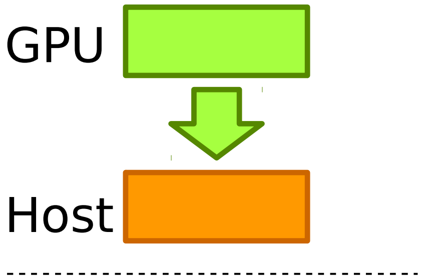


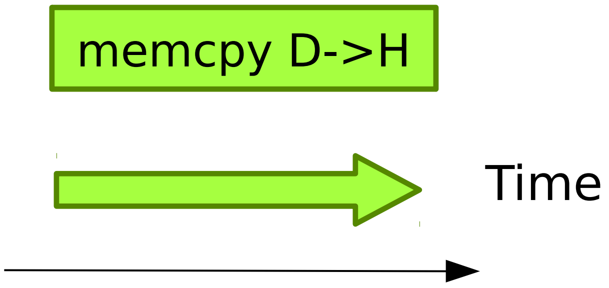


Host memory allocated with **`malloc`** is staged through a pinned memory buffer managed by the CUDA Driver

- 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 IT?

- 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 new memory
  - Using **`cudaHostRegister/cudaHostUnregister`** to pin memory after allocation
- **`cudaMemcpy`** makes automatic use of it
- **`cudaMemcpyAsync`** can be used to issue asynchronous memory copies
- Can be directly accessed from Kernels (zero-copy) – use **`cudaHostGetDevicePointer`**

CUDA STREAMS

- 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 resources: two copy engines, compute resources)

CUDA STREAMS – HOW TO USE THEM?

- Create/Destroy

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

- Launch

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

- Synchronize

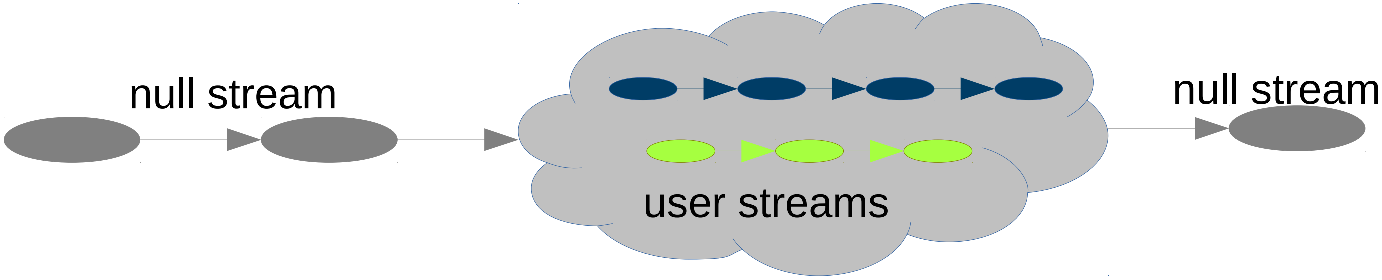
```
cudaStreamSynchronize( stream );
```

CUDA STREAMS – THE DEFAULT (NULL) STREAM

- Kernel launches are always asynchronous
  - Which stream is used here ?

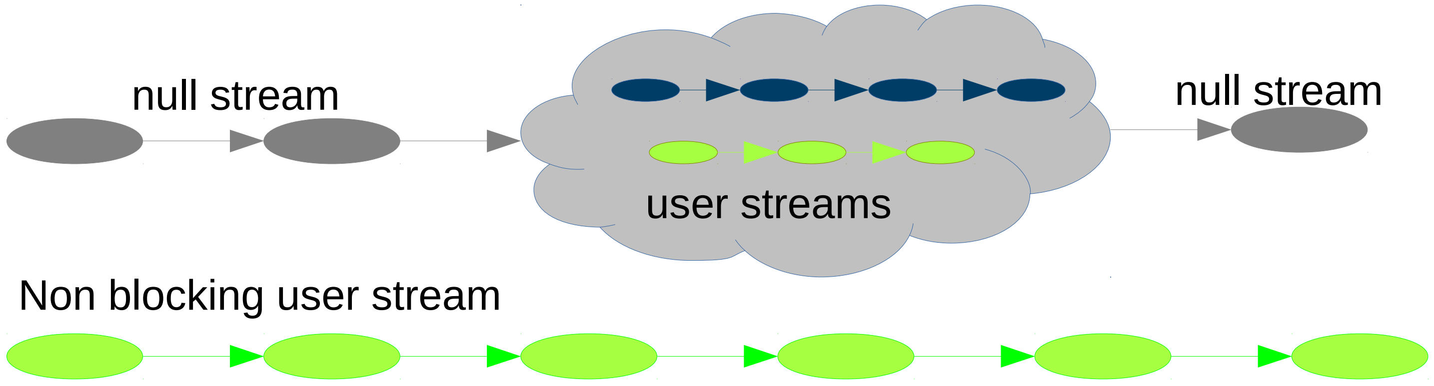
```
my_kernel<<<grid,block>>> (...);
```

  - The default (null) stream



CUDA STREAMS – THE DEFAULT (NULL) STREAM

- **Watch out for false dependencies !**
  - The default stream waits for work in all other streams which do not have the `cudaStreamNonBlock` flag set



User streams with the `cudaStreamNonBlocking` flag set can execute concurrently to stream 0

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 THEM?

Create/Destroy

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

Record

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

Query

```
cudaEventQuery( event );
```

Synchronize

```
cudaEventSynchronize( event );
```

Timing

```
cudaEventElapsedTime( &time, start, end );
```

CUDA EVENTS – EXAMPLE FOR KERNEL TIMING

KERNEL TIMING

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

//Host can do other work

//Get runtime of my_kernel in ms
float runtime = 0.0f;
cudaEventSynchronize ( endEvent );
cudaEventElapsedTime ( &runtime, startEvent, endEvent );
```

CALLING CUBLAS

HOW TO USE CUBLAS FUNCTION

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

//Initialize cuBLAS
cublasCreate(&handle);

//Set cuBLAS exectuion stream
cublasSetStream(handle, stream);

//Call SAXPY
cublasSaxpy(handle, n, &alpha, x, 1, y, 1);
...
//Free up resources
cublasDestroy(handle);
```

THE COMMAND LINE PROFILER NVPROF

- Simple launcher to get profiles of your application
- Profiles CUDA Kernels and API calls

HOW TO USE COMMAND LINE PROFILER

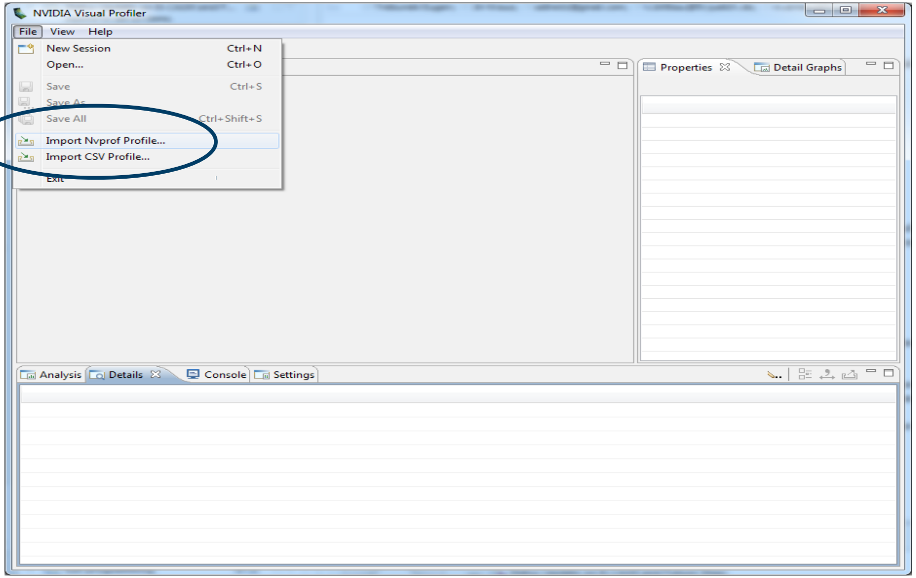
```
    > nvprof ./jacobi

======== NVPROF is profiling jacobi...
======== Command: jacobi
Jacobi (serial)
[…] snip
======== Profiling result:
 Time(%)      Time   Calls       Avg       Min       Max  Name
   72.14  352.65ms    1000  352.65us  350.48us  354.94us  Jacobi_86_gpu
   26.02  127.23ms    1000  127.23us   93.48us  128.34us  Jacobi_74_gpu
    0.84    4.09ms    1000    4.09us    4.04us    4.36us  Jacobi_96_gpu_red
    0.61    3.00ms    1009    2.97us    2.78us   56.16us  [CUDA memcpy HtoD]
    0.39    1.91ms    1002    1.91us    1.82us   52.41us  [CUDA memcpy DtoH]
```


JÜLICH Forschungszentrum

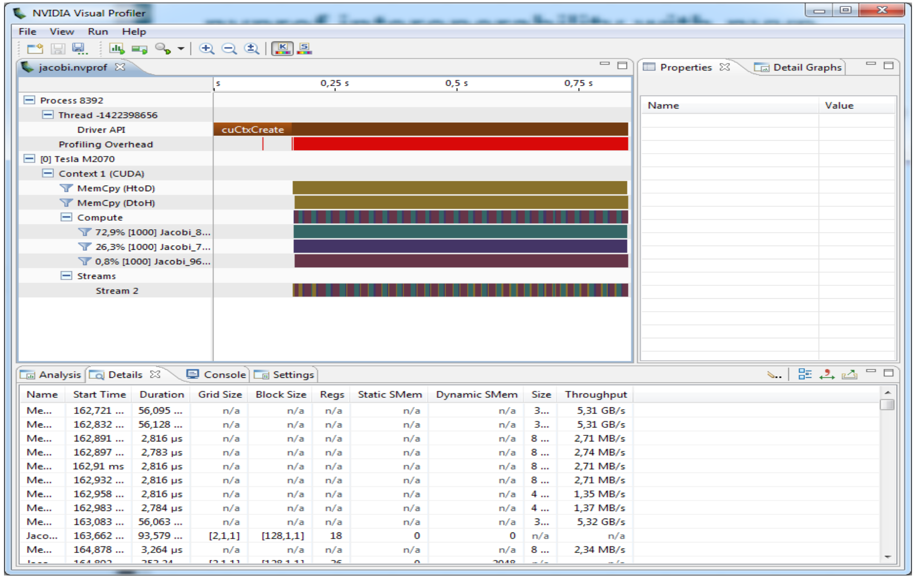
NVPROF INTEROPERABILITY WITH NVVP

- nvprof can write the application profile to nvvp compatible file:

```
nvprof -o jacobi.nvprof ./jacobi
```



Import with nvvp



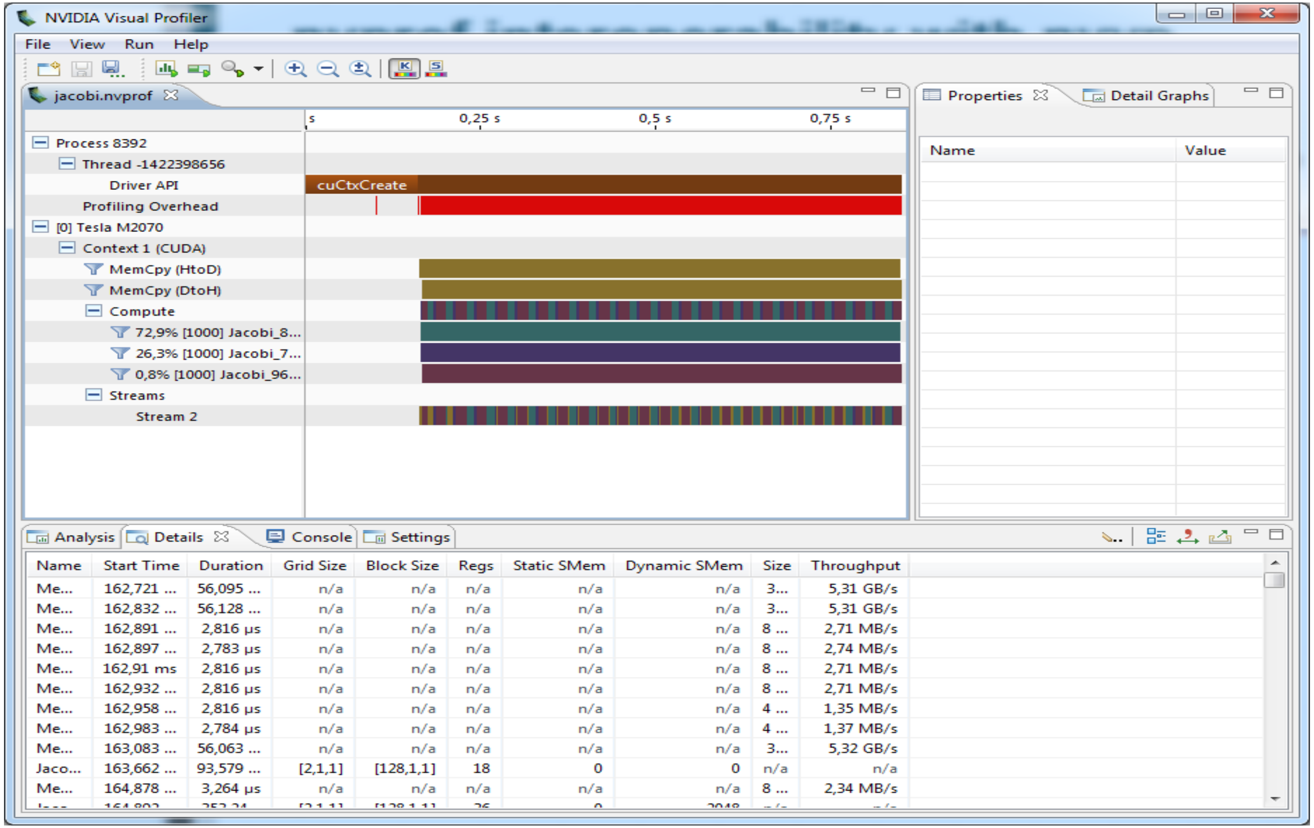
JÜLICH Forschungszentrum

NVPROF IMPORTANT COMMAND-LINE OPTIONS

HOW TO USE COMMAND LINE PROFILER

```
Options:

  -o,  --output-profile <filename>
                            Output the result file which can be imported later
                            or opened by the NVIDIA Visual Profiler.
 --events <event names>
                            Specify the events to be profiled on certain
                            device(s). Multiple event names separated by comma
                            can be specified. Which device(s) are profiled is
                            controlled by the '--devices' option. Otherwise
                            events will be collected on all devices.
                            For a list of available events, use
                            '--query-events'.
--query-events             List all the events available on each device.
-h,  --help                 Print this help information.
```

VISUAL PROFILE - NVVP



JÜLICH Forschungszentrum

EXERCISES: TASK 1

Optimize and overlap host to device and device to host transfers

- Task 1a:
  - Follow TODOs in ***`CUDAStreams/exercises/tasks/task1a.cu`***
    - Allocate host buffers in pinned memory
  - View nvprof profile in nvvp
- Task 1b:
  - Follow TODOs in ***`CUDAStreams/exercises/tasks/task1b.cu`***
    - Create Upload and Download Stream
    - Issue Host to Device and Device to Host Transfer asynchronously in the two new streams.
  - View nvprof profile in nvvp
-

JÜLICH Forschungszentrum

EXERCISES: TASK 2

- Follow TODOs in ***CUDAStreams/exercises/tasks/task2.cu***
  - Set CUBLAS execution stream
  - Call CUBLAS SAXPY
  - Fix position of cudaStreamSynchronize

- Instructions can be found in ***Instructions.rst***
- Solutions in ***CUDAStreams/exercises/solutions***
- Slides can be found in in

  CUDAStreams/slides/CUDAStreams_and_Events.pdf