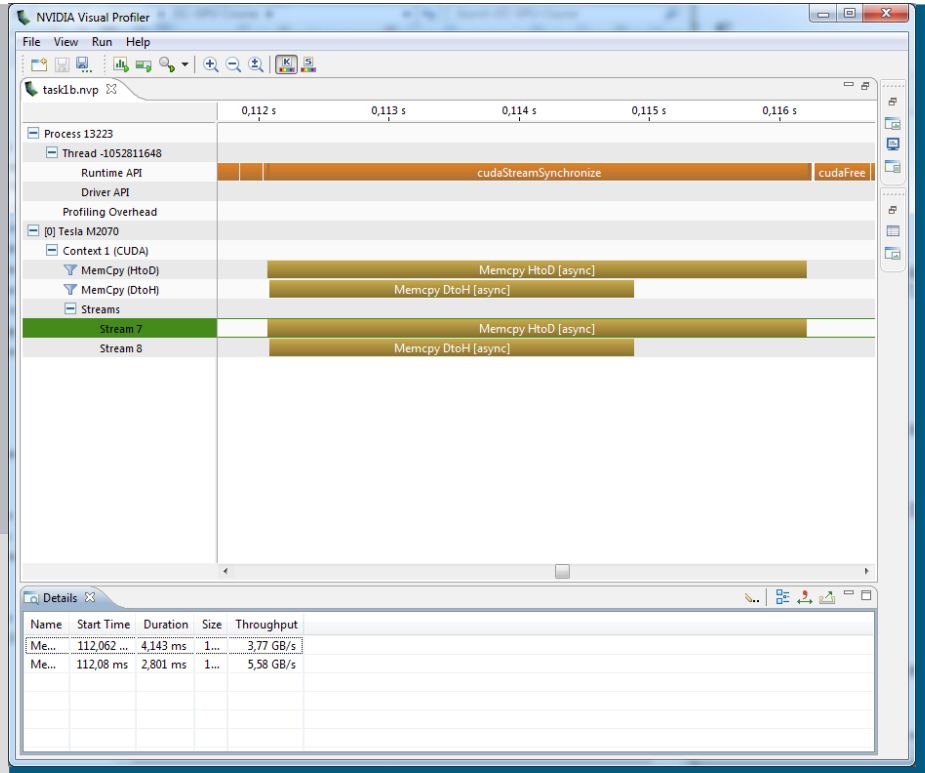



# CUDA Streams, Events and asynchronous memory copies

*Jochen Kreutz*

***GPU Programming@Jülich Supercomputing Centre | Jülich | 25-27 April 2016***

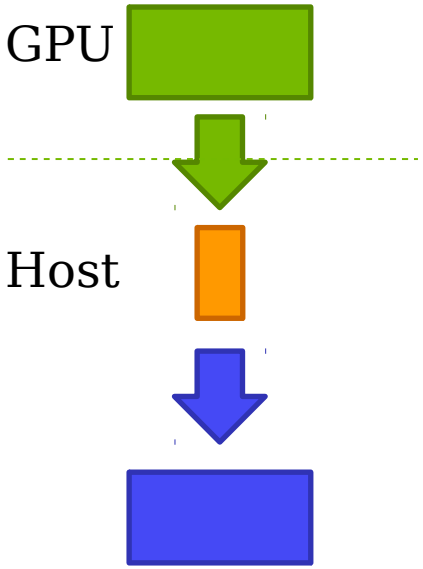
# Overview

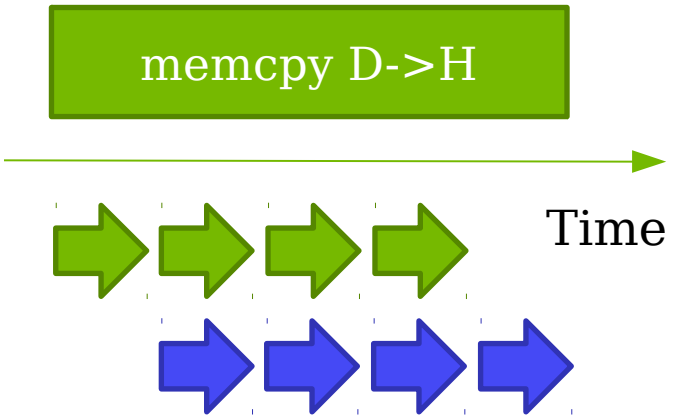
- Pinned (pagelocked) host memory
- Asynchronous and concurrent memory copies
- CUDA streams
  - The default stream and the cudaStreamNonBlocking flag
- CUDA Events
- CUBLAS
- nvprof + nvvp recap

# 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

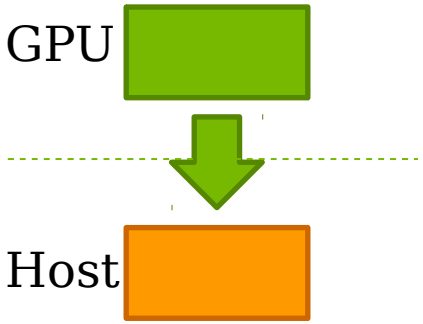


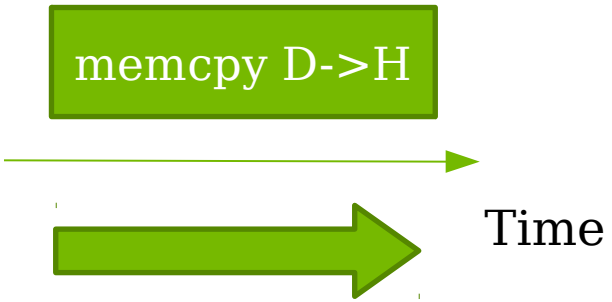




Host memory allocated with **`malloc`** is staged through a pinned memory buffer managed by the CUDA Driver

- 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

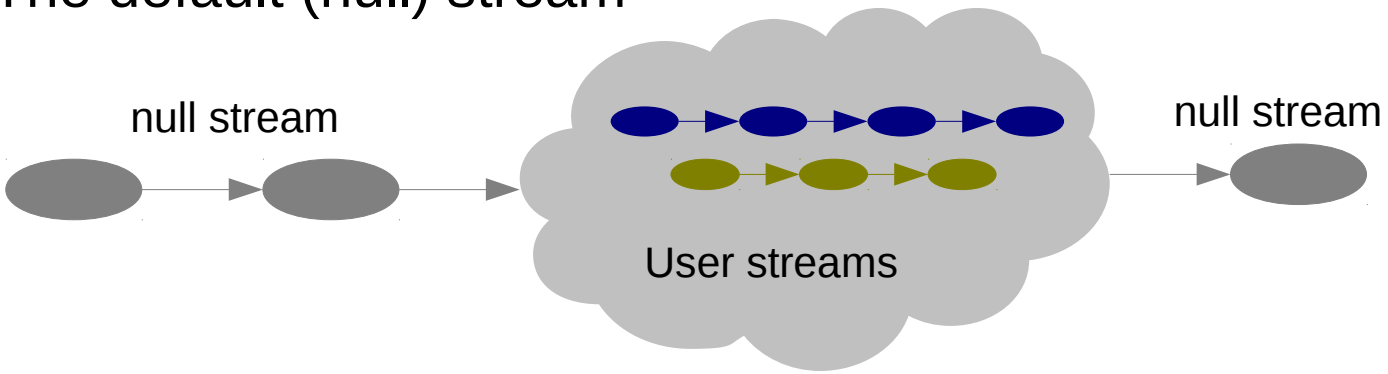
```
cudaStreamSynchronize( stream );
```

# CUDA Streams – The default (null) stream

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

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

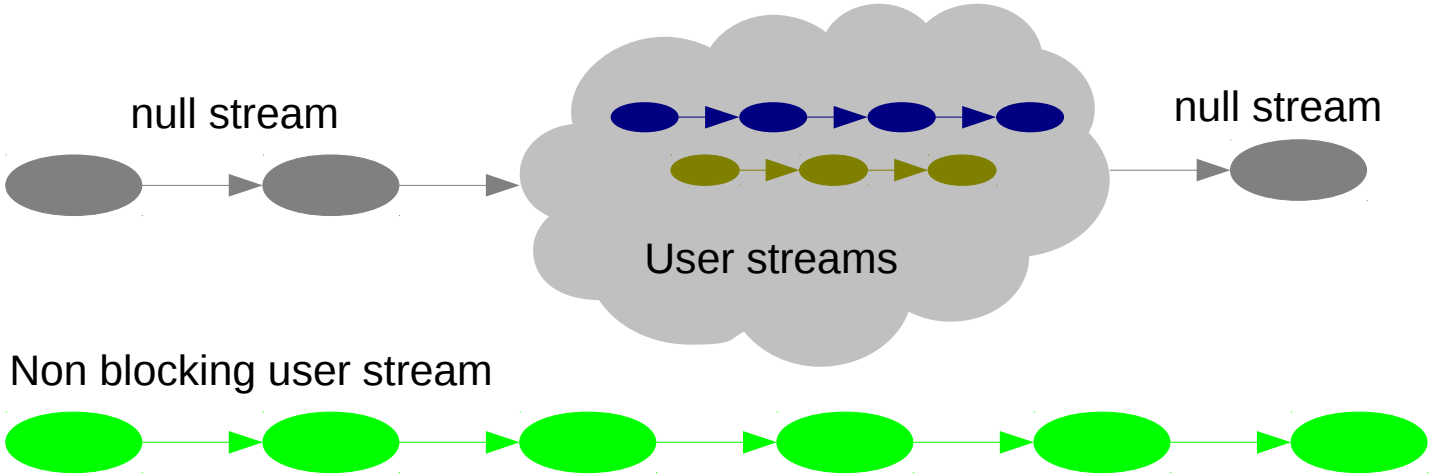
  - The default (null) stream



# CUDA Streams – The cudaStreamNonBlocking

## Watch for false dependencies!

- The default 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 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

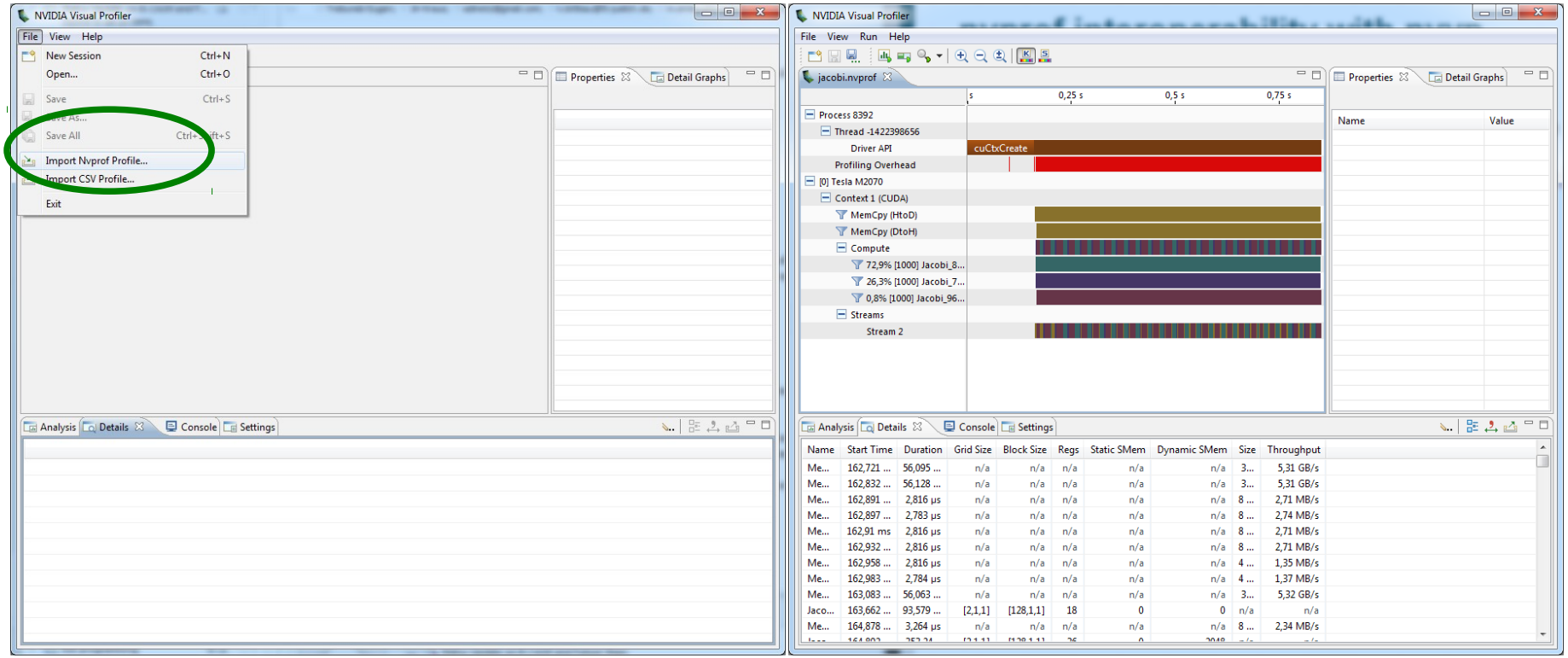
```
    > nvprof ./jacobi

======== 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]
```

# nvprof interoperability with nvvp

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

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

- Import in nvvp

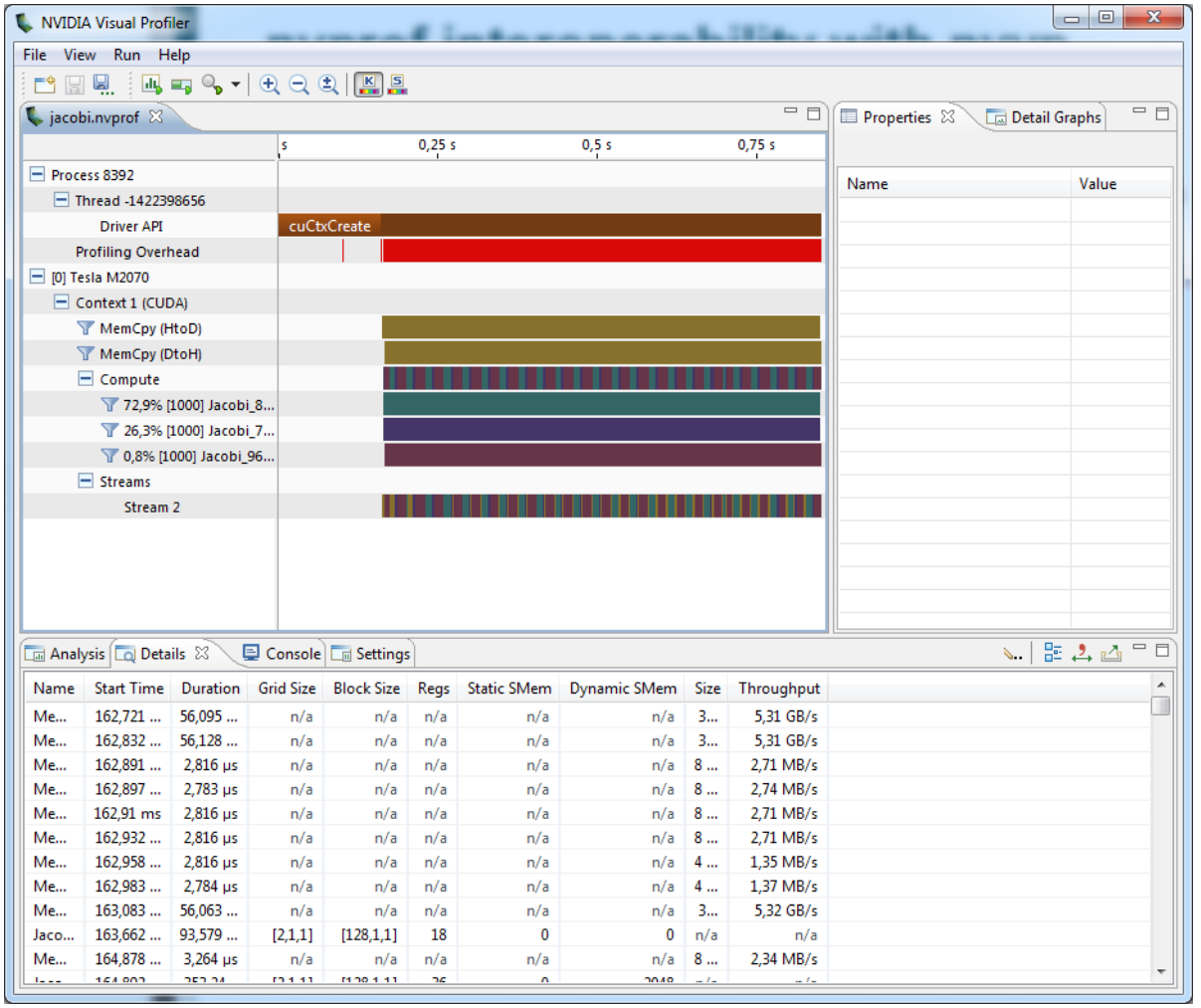


# nvprof important command-line options

## 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.
```

# nvvp


NVIDIA Visual Profiler
File View Run Help
jacobi.nvprof
Properties
Detail Graphs
s 0,25 s 0,5 s 0,75 s
Process 8392
Thread -1422398656
Driver API
cuCtxCreate
Profiling Overhead
[0] Tesla M2070
Context 1 (CUDA)
MemCpy (HtoD)
MemCpy (DtoH)
Compute
72,9% [1000] Jacobi_8...
26,3% [1000] Jacobi_7...
0,8% [1000] Jacobi_96...
Streams
Stream 2
Name Value
Analysis Details Console Settings
Name Start Time Duration Grid Size Block Size Regs Static SMem Dynamic SMem Size Throughput
Me... 162,721 ... 56,095 ... n/a n/a n/a n/a n/a 3... 5,31 GB/s
Me... 162,832 ... 56,128 ... n/a n/a n/a n/a n/a 3... 5,31 GB/s
Me... 162,891 ... 2,816 µs n/a n/a n/a n/a n/a 8 ... 2,71 MB/s
Me... 162,897 ... 2,783 µs n/a n/a n/a n/a n/a 8 ... 2,74 MB/s
Me... 162,91 ms 2,816 µs n/a n/a n/a n/a n/a 8 ... 2,71 MB/s
Me... 162,932 ... 2,816 µs n/a n/a n/a n/a n/a 8 ... 2,71 MB/s
Me... 162,958 ... 2,816 µs n/a n/a n/a n/a n/a 4 ... 1,35 MB/s
Me... 162,983 ... 2,784 µs n/a n/a n/a n/a n/a 4 ... 1,37 MB/s
Me... 163,083 ... 56,063 ... n/a n/a n/a n/a n/a 3... 5,32 GB/s
Jaco... 163,662 ... 93,579 ... [2,1,1] [128,1,1] 18 0 0 n/a n/a
Me... 164,878 ... 3,264 µs n/a n/a n/a n/a n/a 8 ... 2,34 MB/s

# 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
- Solutions in *CUDAStreams/exercises/solutions*
- Slides are in

*CUDAStreams/slides/CUDAStreams_and_Events.pdf*

# Task 2:

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

  *CUDAStreams/slides/CUDAStreams_and_Events.pdf*