center for
excellence in parallel
programming

CUDA

Georges-Emmanuel Moulard
Paul Karlshöfer

Bull
atos technologies

RECAP of CUDA Fundamentals

- Addressed subjects so far:
  - Hardware topology
  - Synchronous memory management
  - Kernel execution

  - Code porting -> from C/C++/Fortran -> CUDA (nbody simulation example)

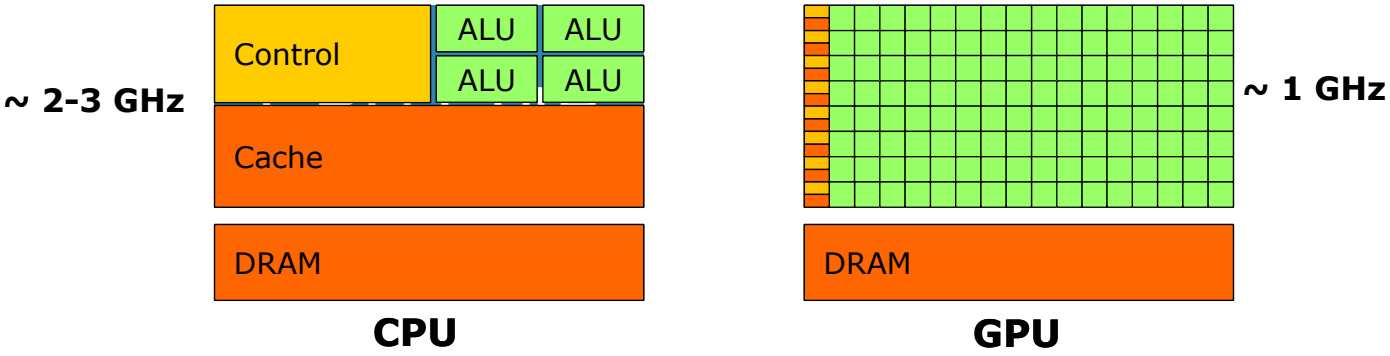
  - Memory modes (pageable, pinned, mapped, *managed*)
  - Profiling with nvprof (or nsys)
  - CUDA streams
  - CUDA events

So, what's left?

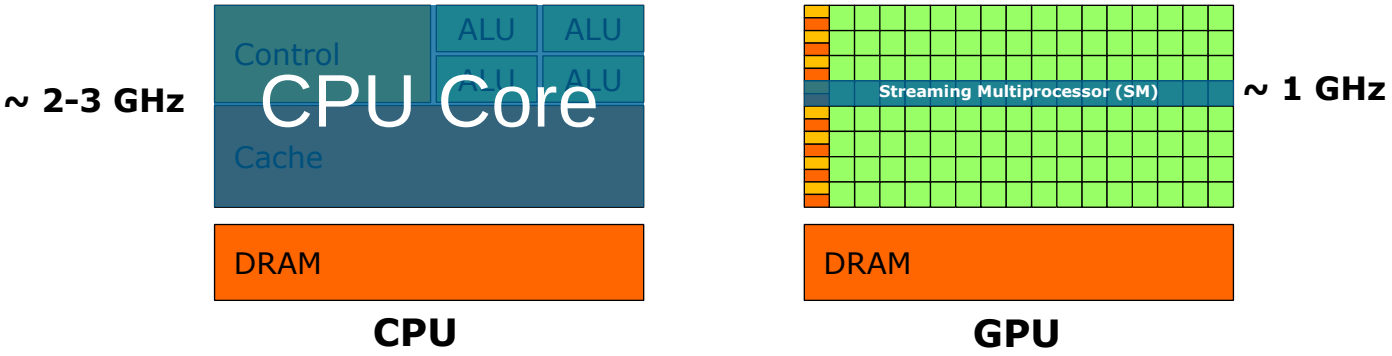
- Debugging
- Kernel optimization
  - Warps
  - Effects of global memory access
- On-chip Memory (shared, constant, L2, L1)
  - General usage
  - Banks
- Compilation for specific hardware
- Multi-GPU

- And some other features of the CUDA model

RECAP -



RECAP -



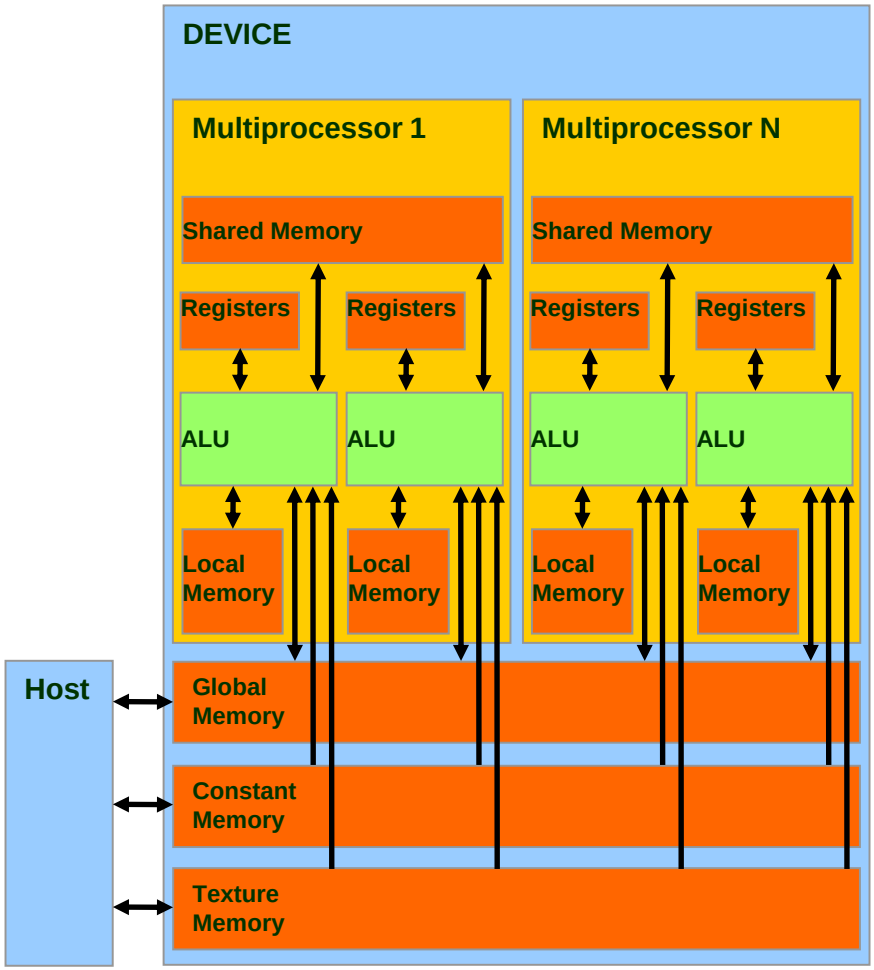

```
+-----------------------------------------------------------------------------+
| Nvidia-SMI 340.29     Driver Version: 340.29                                |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla K80           On   | 0000:04:00.0     Off |                    0 |
| N/A   66C    P0   108W / 149W |    240MiB / 11519MiB |     40%   E. Process |
+-------------------------------+----------------------+----------------------+
|   1  Tesla K80           On   | 0000:05:00.0     Off |                    0 |
| N/A   55C    P0   136W / 149W |    260MiB / 11519MiB |     68%   E. Process |
+-------------------------------+----------------------+----------------------+
|   2  Tesla K80           On   | 0000:86:00.0     Off |                    0 |
| N/A   47C    P0   123W / 149W |    252MiB / 11519MiB |     61%   E. Process |
+-------------------------------+----------------------+----------------------+
|   3  Tesla K80           On   | 0000:87:00.0     Off |                    0 |
| N/A   57C    P0   115W / 149W |    242MiB / 11519MiB |     40%   E. Process |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Compute processes:                                               GPU Memory |
| GPU       PID  Process name                                      Usage      |
+-----------------------------------------------------------------------------+
```

RECAP - environment

- $> export CUDA_VISIBLE_DEVICES=0,1,2,…,N

```
export CUDA_HOME=/opt/cuda/9.2
export PATH=$CUDA_HOME/bin:$PATH
export LD_LIBRARY_PATH=$CUDA_HOME/lib64:$LD_LIBRARY_PATH
export CUDA_INC=$CUDA_HOME/include
```

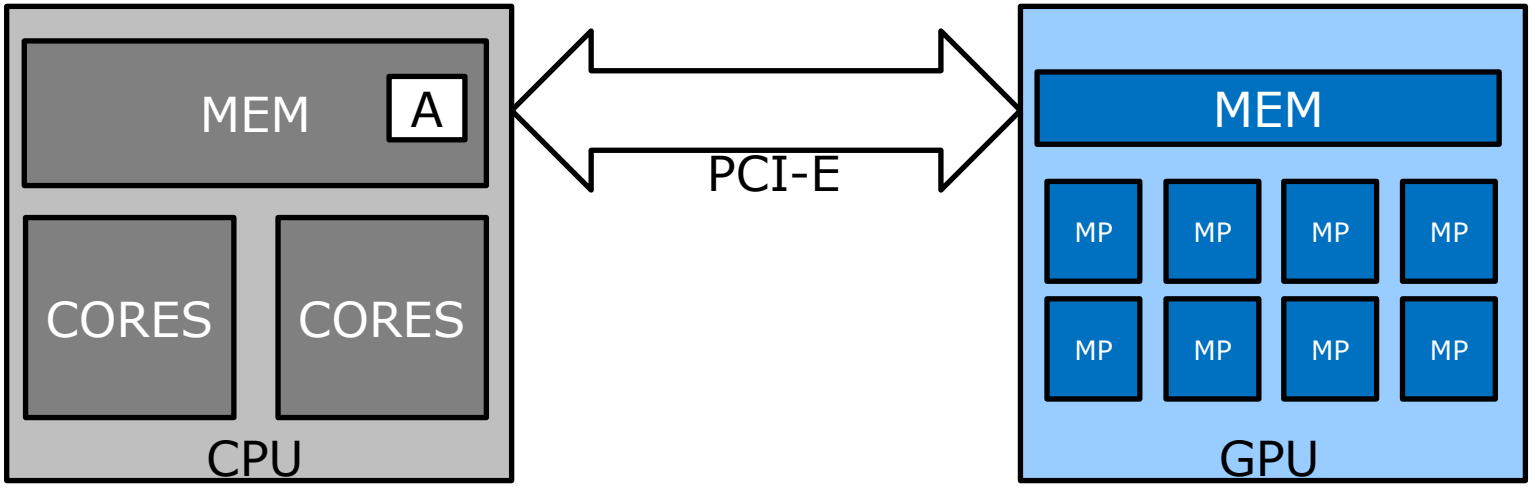

RECAP – topologic structure of a GPU



RECAP - How to compute on GPU

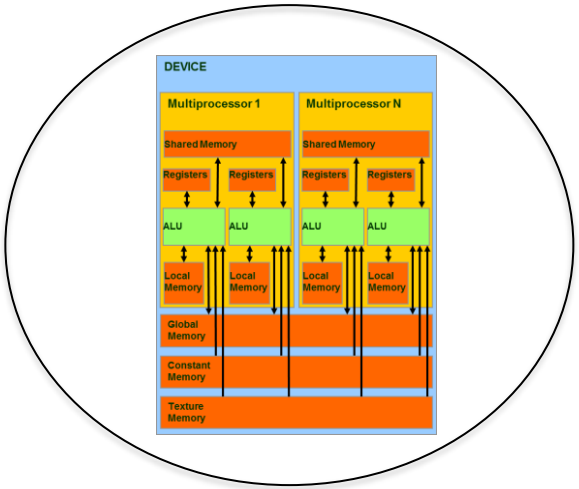
- **5 steps to offload computation on the GPU:**

- (1) Memory Allocation : **cudaMalloc(&B, …)**
- (2) H2D transfer : **cudaMemcpy(B, A, …)**
- (3) Execute : **kernel<<< … >>>(…)**
- (4) D2H transfer : **cudaMemcpy(A, B, …)**
- (5) Free Memory : **cudaFree(B)**



RECAP - Kernel Execution

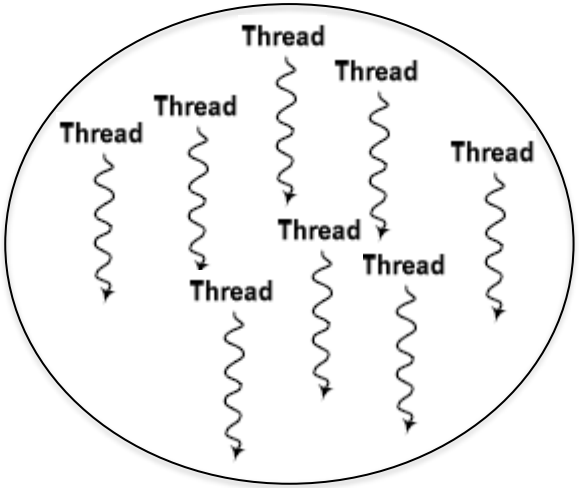
A hardware abstract view



GPU functions

```
__global__ void myFunction( ... )
{
  ...
}
```

Pools of threads

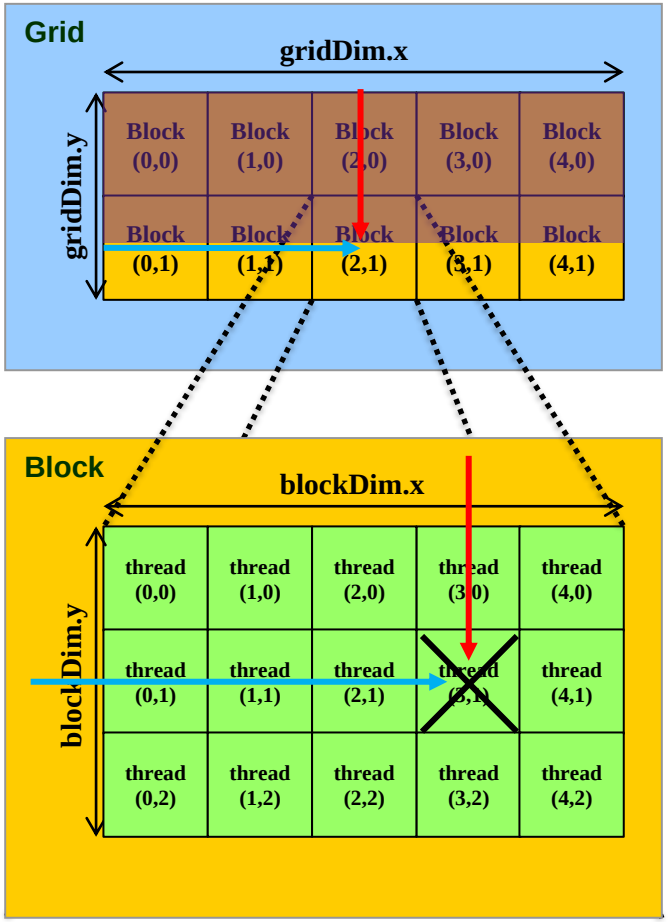


RECAP - 2D Grid Linearization

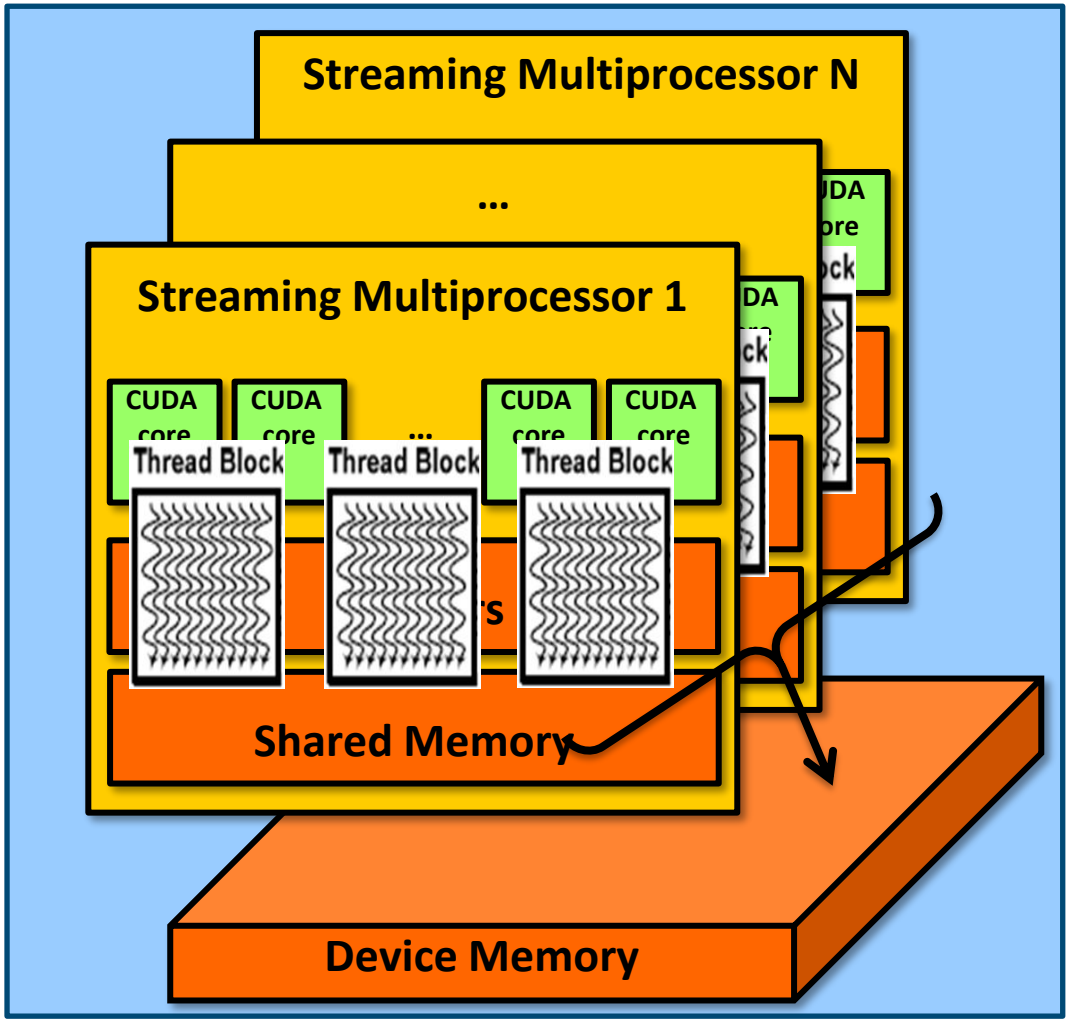
- 2D THREAD INDEXING

index_x = threadIdx.x + blockIdx.x * blockDim.x
index_y = threadIdx.y + blockIdx.y * blockDim.y

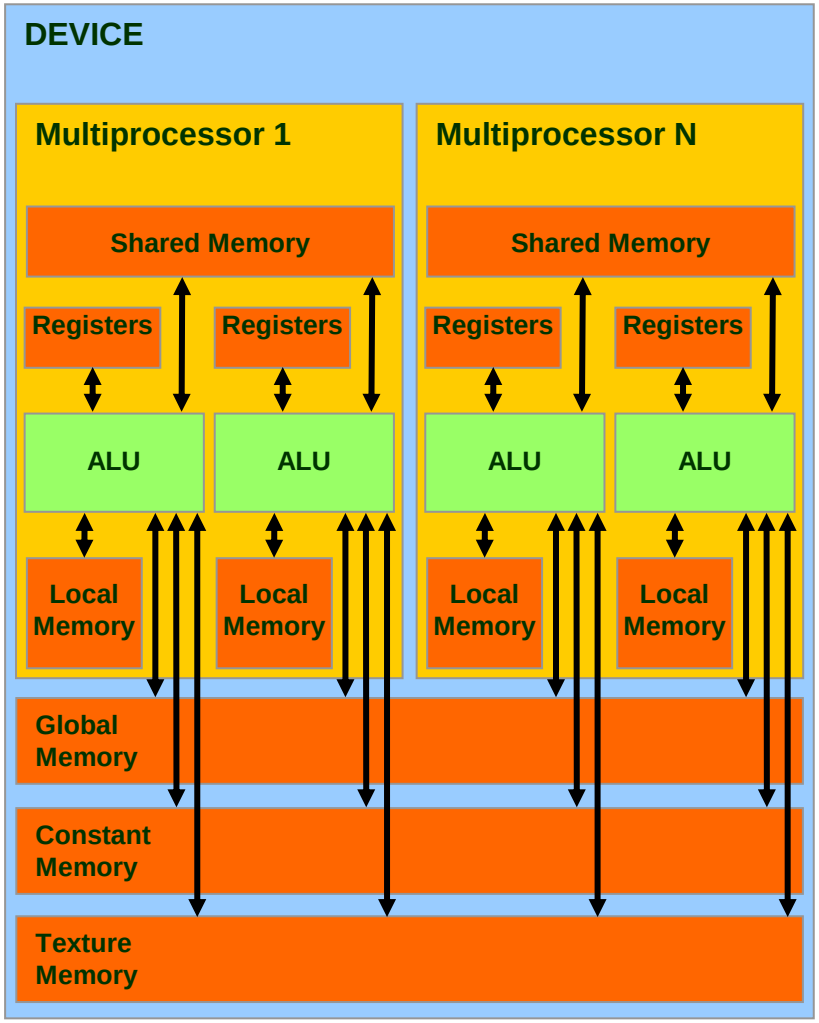
index = index_x + index_y * gridDim.x * blockDim.x

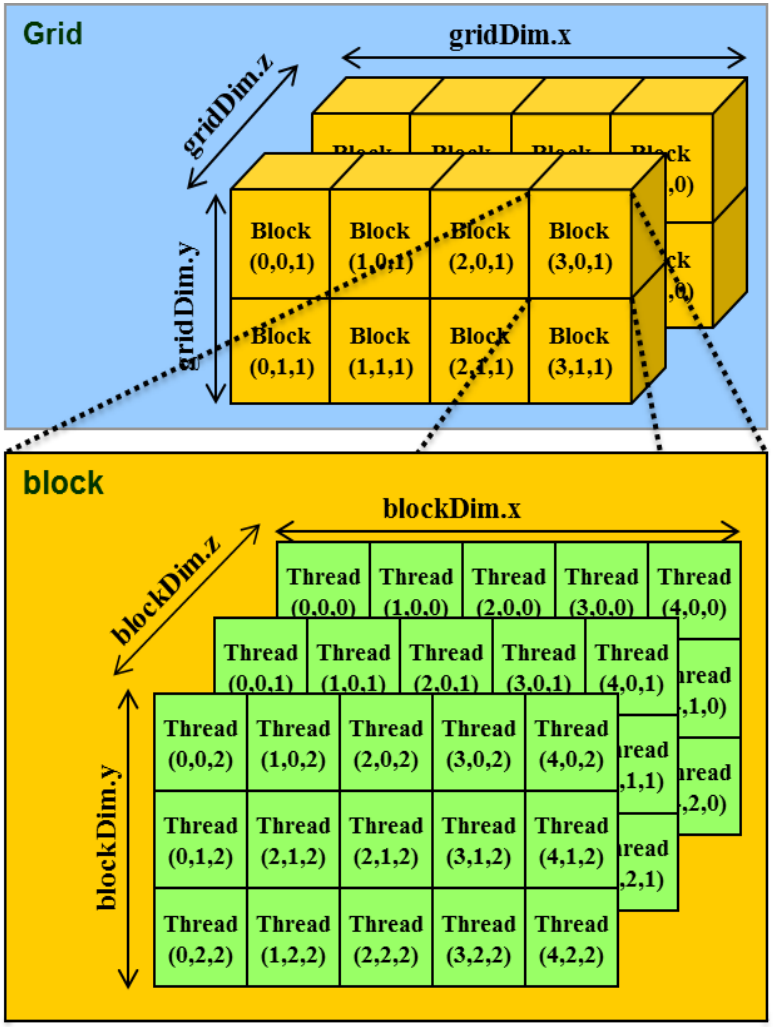


RECAP - scheduling



RECAP - Memory Hierarchy





RECAP – CUDA streams

```
cudaStream_t stream[2];
//streams creation
for ( int i = 0; i < 2; ++i)
  cudaStreamCreate (& stream[i]);

float * hostPtr ;
cudaMallocHost(...); //page-locked memory allocation

cudaMemcpyAsync ( ... , cudaMemcpyHostToDevice , stream[0]);
cudaMemcpyAsync ( ... , cudaMemcpyHostToDevice , stream[1]);

kernel1 <<<100, 512 , 0, stream[0]>>>(...)
cudaStreamSynchronize(stream[0]);
kernel2 <<<100, 512 , 0, stream[1]>>>(...)

cudaMemcpyAsync ( ... , cudaMemcpyDeviceToHost , stream[0]);
cudaMemcpyAsync ( ... , cudaMemcpyDeviceToHost , stream[1]);

cudaDeviceSynchronize();

//streams destruction
for ( int i = 0; i < 2; ++i)
  cudaStreamDestroy (stream [i]);
```

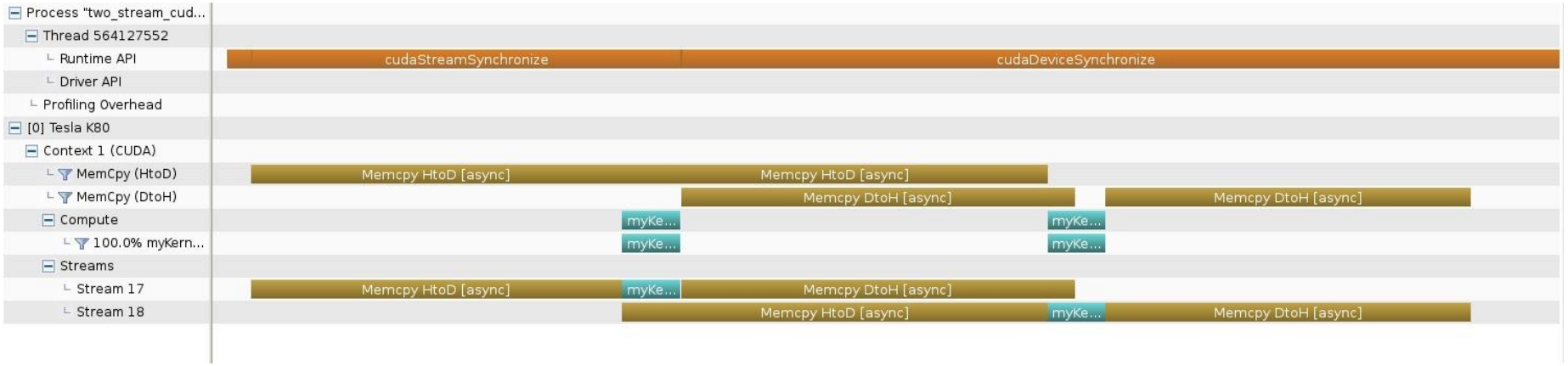
RECAP – CUDA streams

- CUDA stream functions like a queue
- Multiple queues can run in parallel

- Allows for async memory operations
- Allows for multiple kernels to run in parallel

- Special synchronization rules to the default /NULL stream!

RECAP – CUDA streams



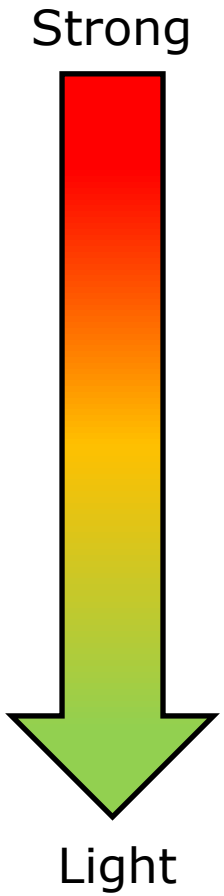
RECAP – CUDA Events

- Allows for light weight synchronization
- Allows for sync across different streams

- Timing

Reminder: Synchronization

- **cudaDeviceSynchronize()**
  - Synchronize everything
    - Blocks host until all issued CUDA calls are complete
- **cudaStreamSynchronize(stream)**
  - Synchronize host with regard to a specific stream
    - Blocks host until all issued CUDA calls in stream are complete
- Synchronize host or devices using **events**
  - allow to synchronize several streams



RECAP – Data transfer Optimizations

- …

- *Check out the handout for function signatures*

Copyright

Thanks

For more information please contact:
Georges-Emmanuel Moulard
M+ 33 6 85529054
georges-emmanuel.moulard@atos.net

29-10-2018

Bull
atos technologies