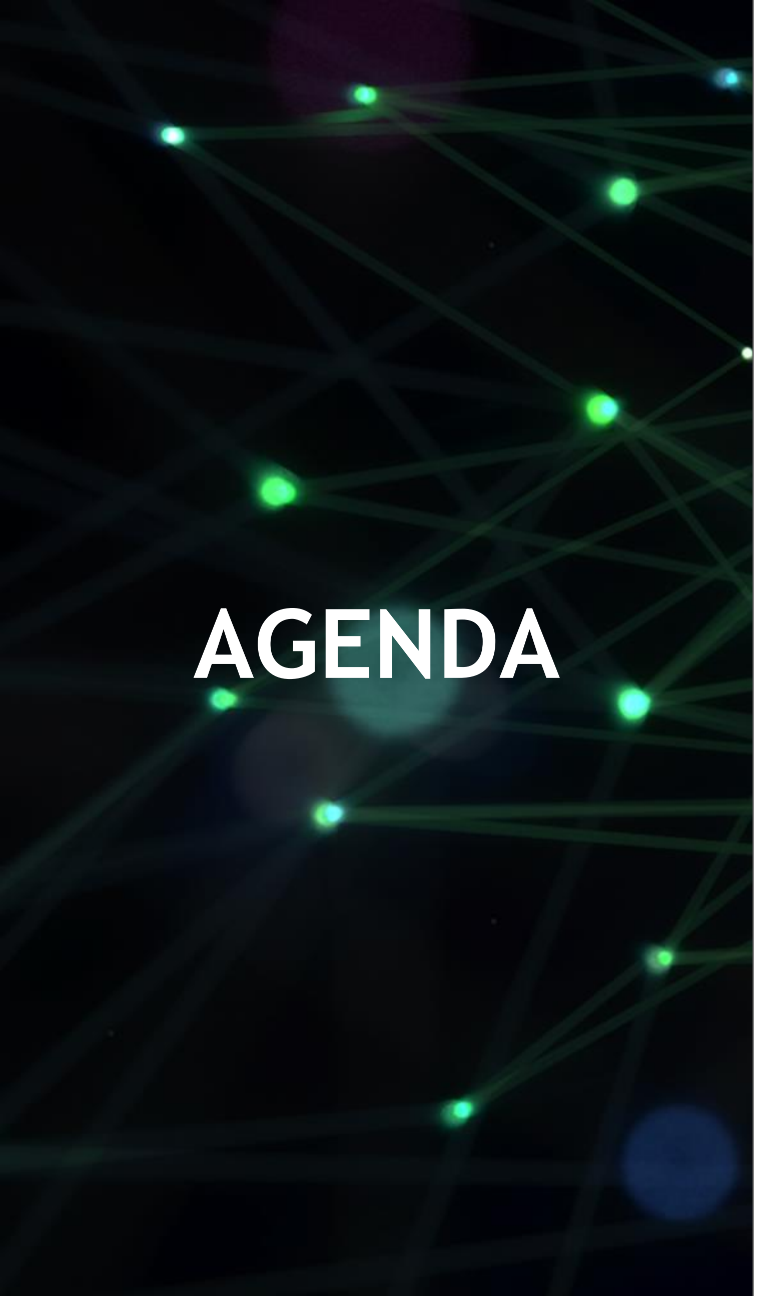
BRINGING GROMACS UP-TO-SPEED ON MODERN MULTI-GPU SYSTEMS

Alan Gray and Jon Vincent, GTC 2019

ACKNOWLEDGEMENTS

- We are very grateful to the core Gromacs development team in Stockholm for the ongoing collaboration, in particular:
  - Erik Lindahl, Stockholm University/SciLifeLab/KTH
  - Mark Abraham, SciLifeLab/KTH
  - Szilard Pall, KTH/PDC
  - Berk Hess, SciLifeLab/KTH
  - Artem Zhmurov, KTH/PDC
- The EU BioExcel Center of Excellence for Biomolecular Research supports this collaboration.
- The results presented here involve NVIDIA’s prototype developments. We are now working with the above team to integrate these into the main Gromacs branch, including further improvements.



AGENDA

- Introduction
- A high-level overview of developments
- Performance results
- Development details
- Attacking small problem sizes with CUDA Graphs

mini-presentation:
“Getting Started With CUDA Graphs”

INTRODUCTION

INTRODUCTION

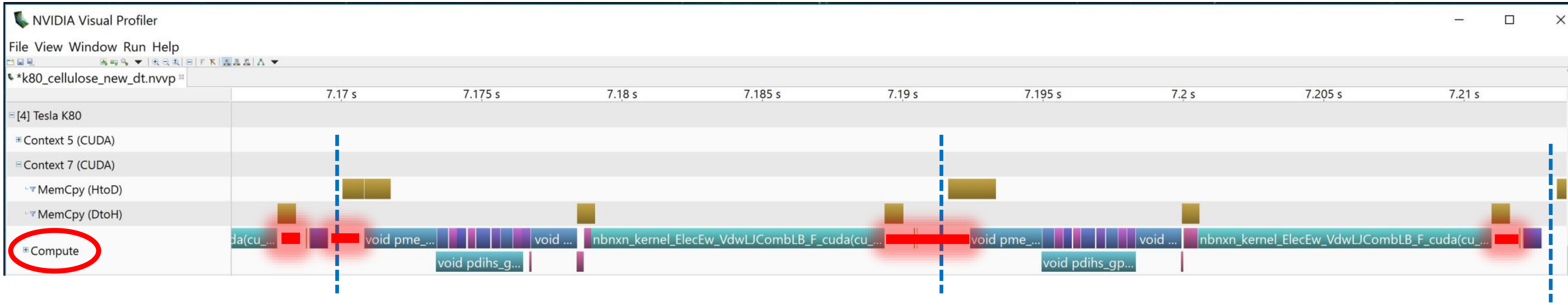
- Gromacs, a simulation package for biomolecular systems, is one of the most highly used HPC applications globally.
- It evolves systems of particles using the Newtonian equations of motion:
    - **Forces** between particles dictate their movement (e.g. two positively charged ions will repel).
- Calculating forces is most expensive part of simulation - all pairs of particles in the simulation can potentially interact. Forces get weaker with distance, but long-range forces still must be accounted for.

INTRODUCTION

- Force calcs typically fall into three classes in Gromacs:
  - **Non-bonded forces**: (short range) - particles within a certain cutoff range interact directly
  - **PME**: long-range forces accounted for through a “Particle Mesh Ewald” scheme, where Fourier transforms are used to perform calculations in Fourier space, which is much cheaper than calculating all interactions directly in real space
  - **Bonded forces**: required due to specific behaviour of bonds between particles, e.g. the harmonic potential when two covalently bonded atoms are stretched
- These are all now accelerated, most recently the addition of GPU bonded forces in Gromacs 2019 (evolved through prototype work by NVIDIA). But we still have a problem….
  - …force calcs are now so fast on modern GPUs that other parts are now very significant, especially when we wish to utilize multiple GPUs.
- We will describe work to port all significant remaining computational kernels to the GPU, and to perform the required Inter-GPU communications using peer-to-peer memory copies, such that the GPU is exploited throughout and repeated PCIe transfers are avoided.

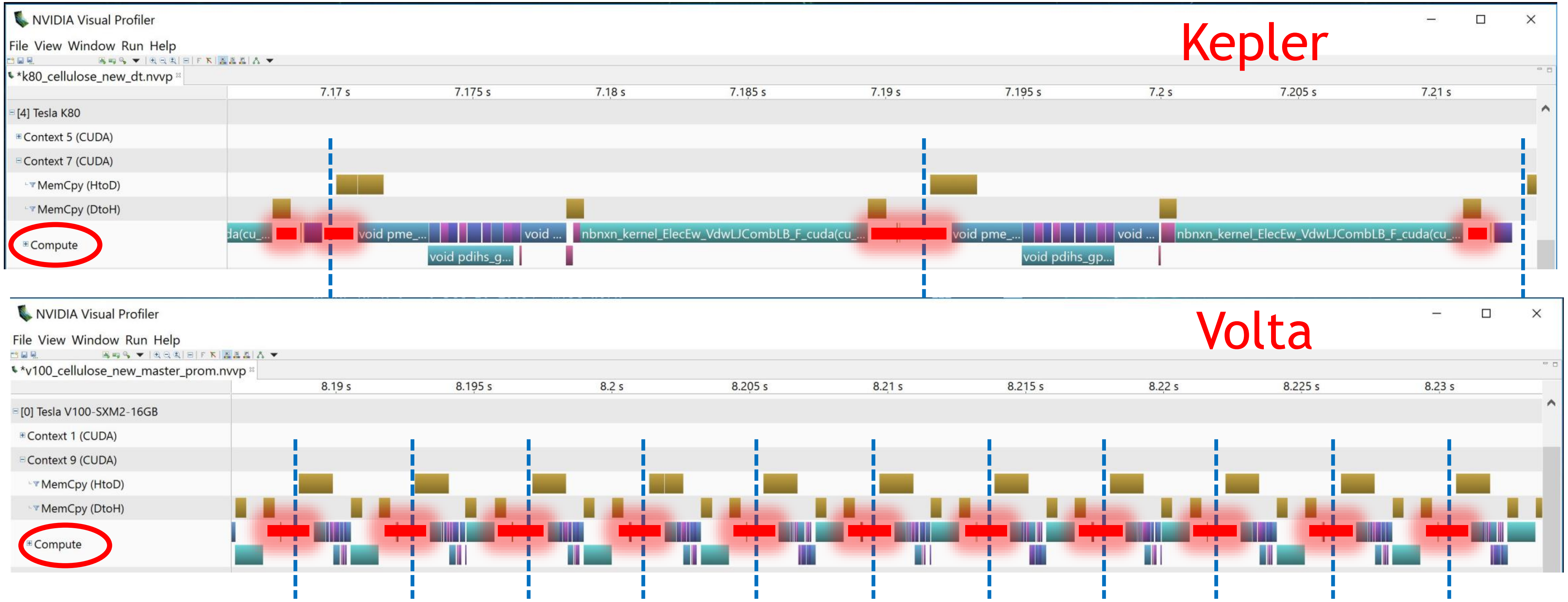
A HIGH LEVEL OVERVIEW OF DEVELOPMENTS

GROMACS ON OLD KEPLER ARCHITECTURE



- On old architectures such as Kepler, force calculations are very dominant and other overheads are dwarfed.
- ~400K atom "Cellulose" case.
- 🟥 : GPU Idle time

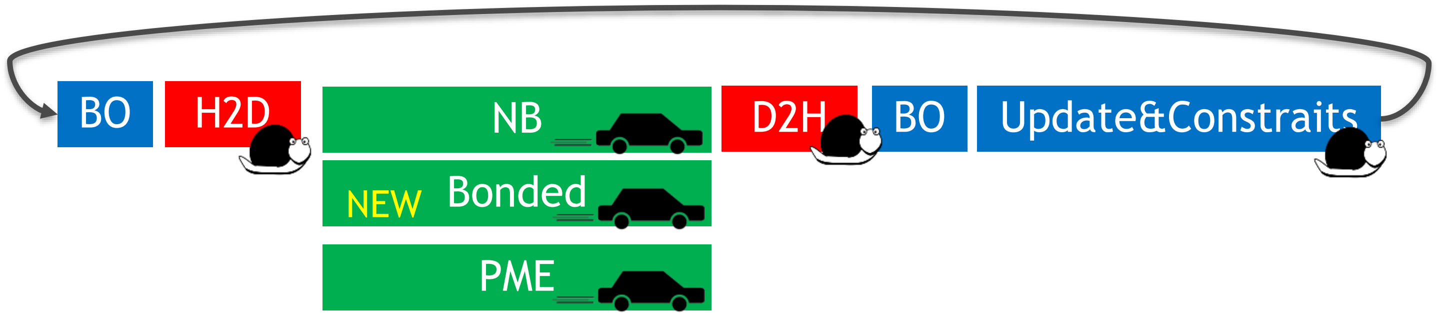
VOLTA VS KEPLER

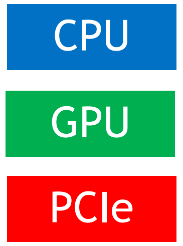


- But on new architectures such as Volta, force kernels are so fast that other overheads are very significant.
  - The timescales are aligned in the above profiles

THE PROBLEM

Single GPU

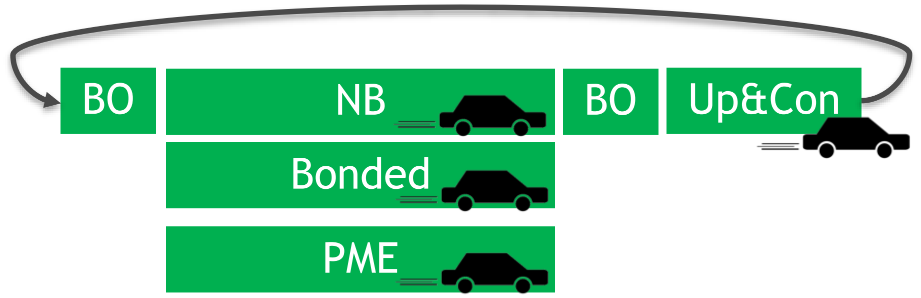


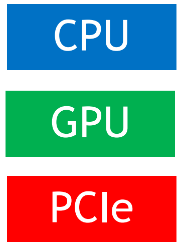


BO = Buffer Ops

THE SOLUTION

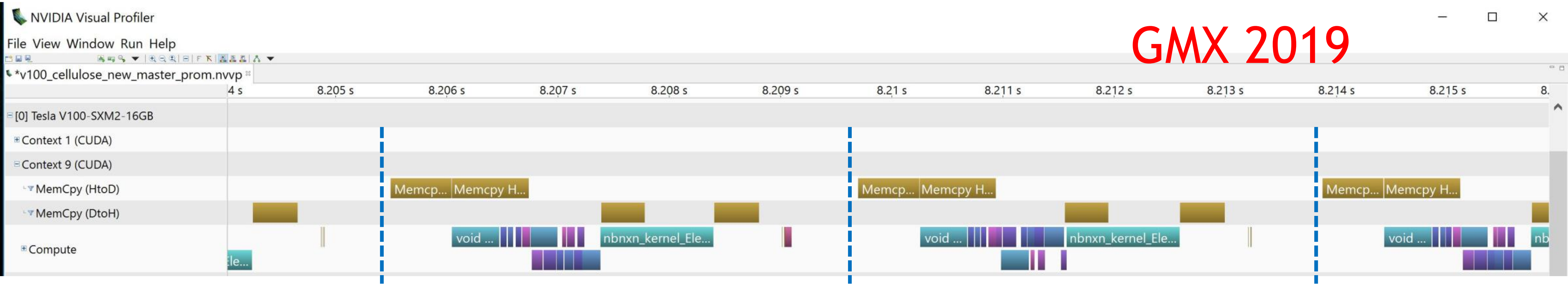
Single GPU

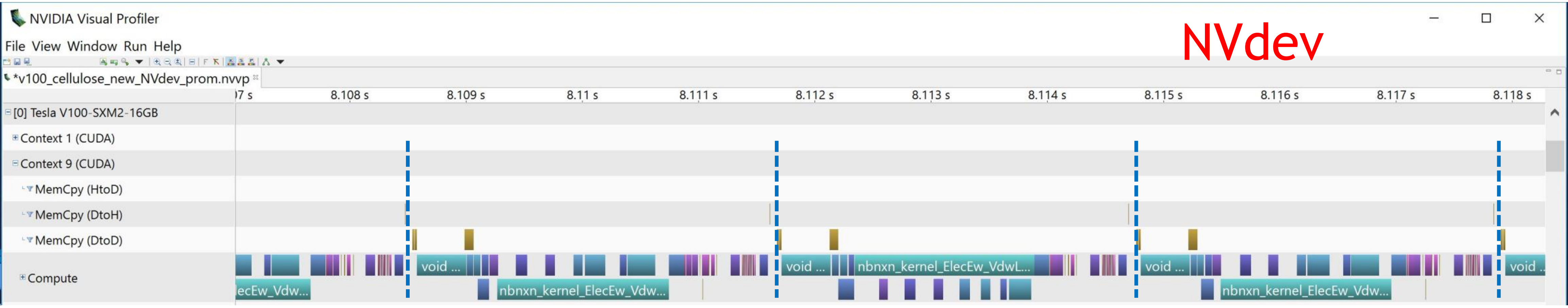




BO = Buffer Ops

SINGLE GPU: NEW DEVELOPMENT

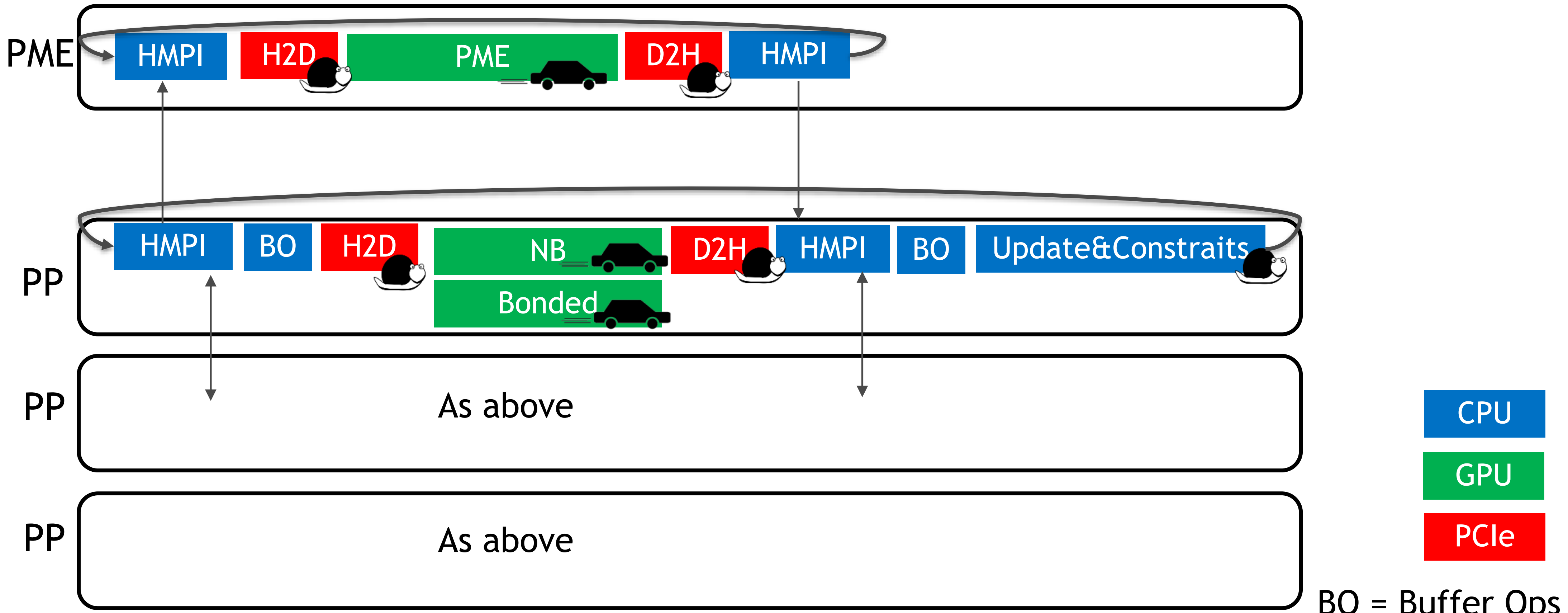




- Aligned timescales

THE PROBLEM

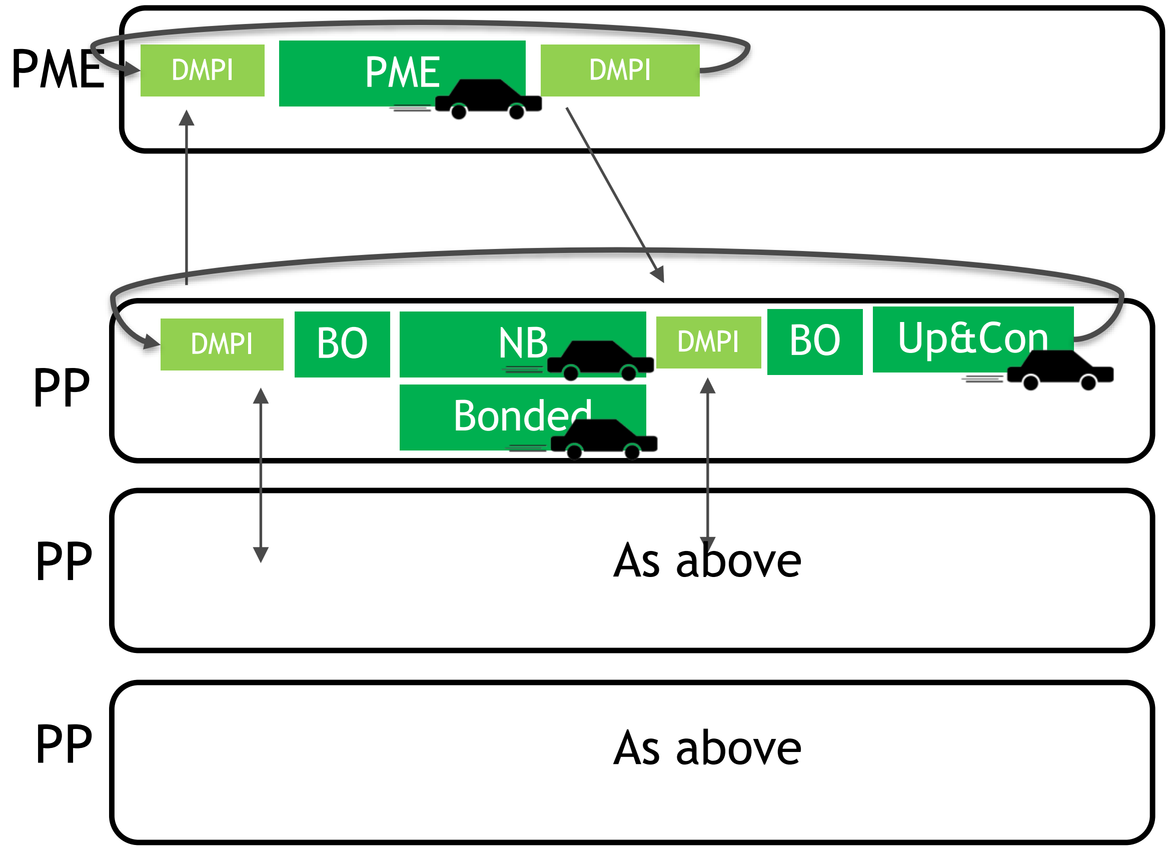
Multi (4X) GPU



BO = Buffer Ops
HMPI = Host MPI

THE SOLUTION

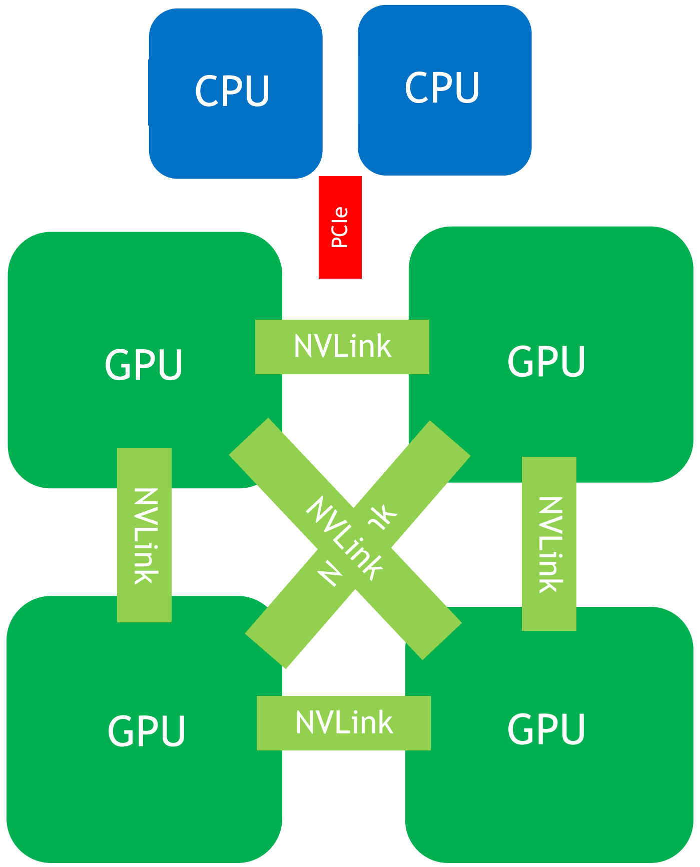
Multi (4X) GPU



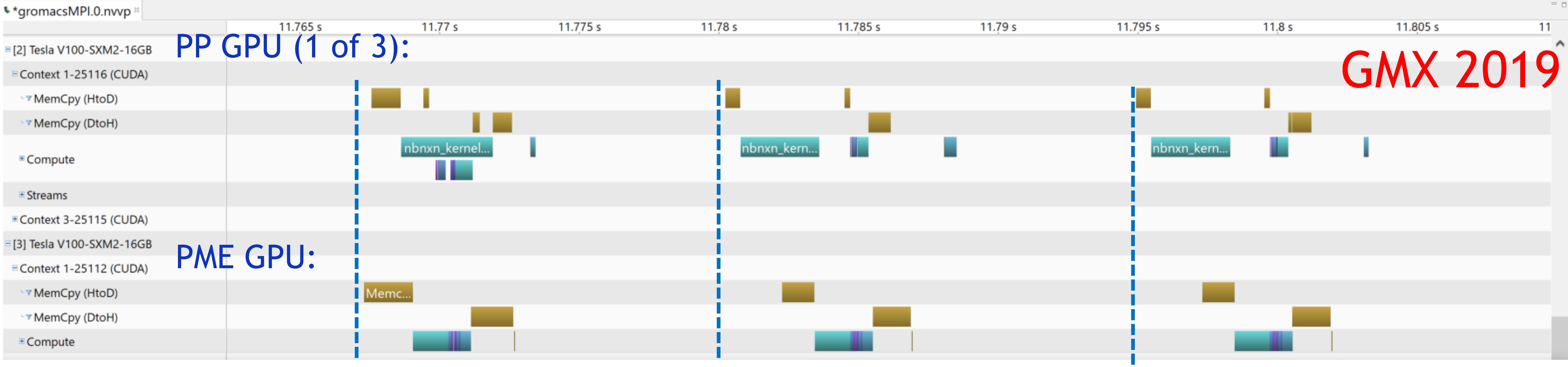
🟩	GPU
🟩	NVLink

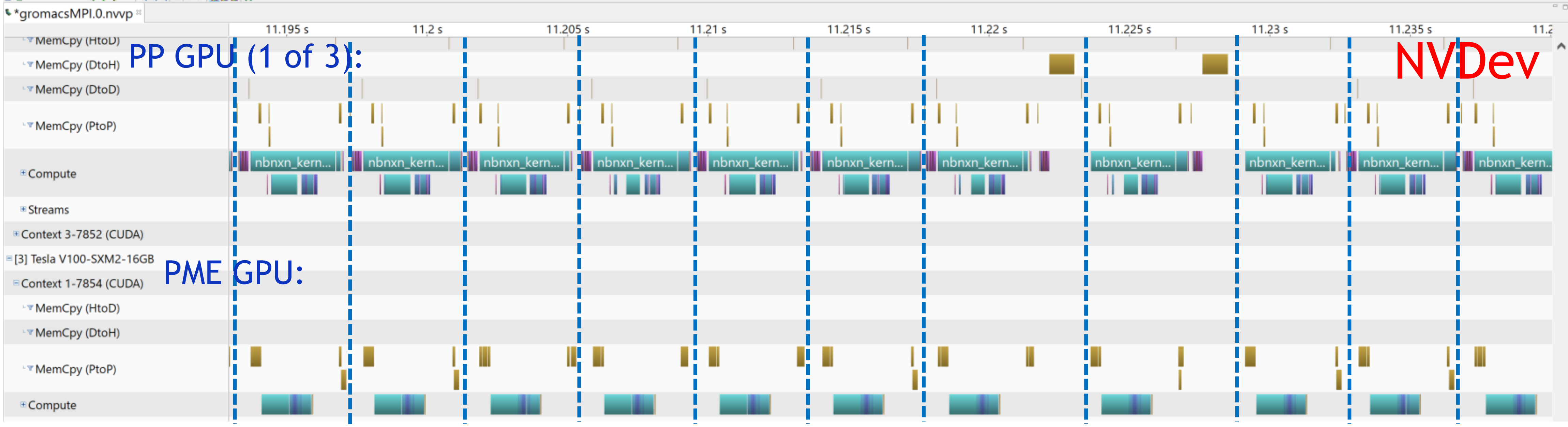
BO = Buffer Ops
DMPI = Device MPI

MULTI-GPU



- For our multi-GPU experiments we use 4 x V100 SXM2 GPUs fully-connected with NVLink, plus 2xCPU.





- Aligned timescales. STMV (~1M atom) case.

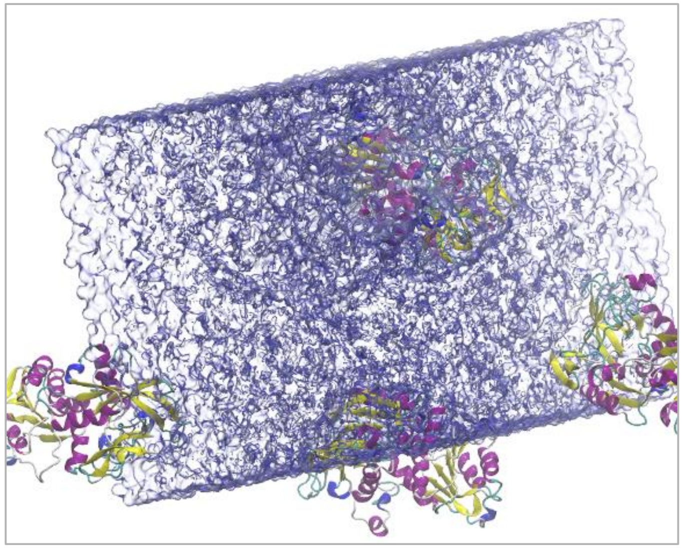
DEVELOPMENT WORKFLOW

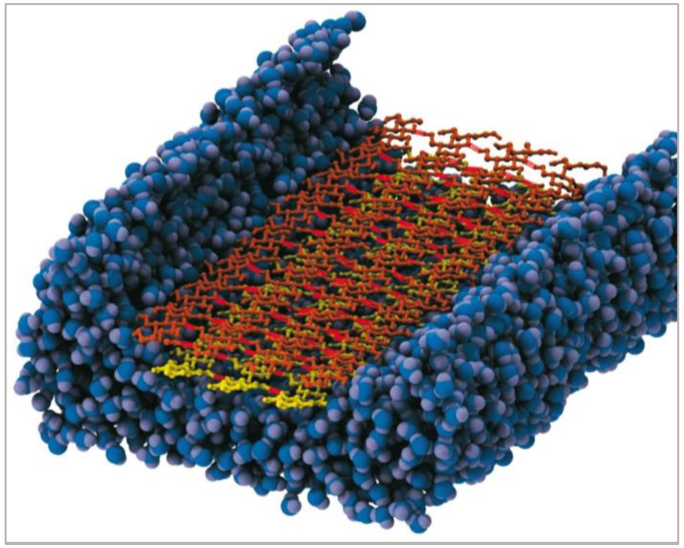
1. Develop a prototype branch of Gromacs
   - Aim to support most commonly used simulation scenarios
   - Demonstrate performance benefits for real test case
   - Sandbox branch of Gromacs gerrit repo: sandbox-puregpu
   - Not designed as a fork suitable for production work
2. Upstream developments into main Gromacs master branch
   - In collaboration with core Gromacs developers
   - Major effort required to refactor and integrate in a robust manner
   - Further performance improvements

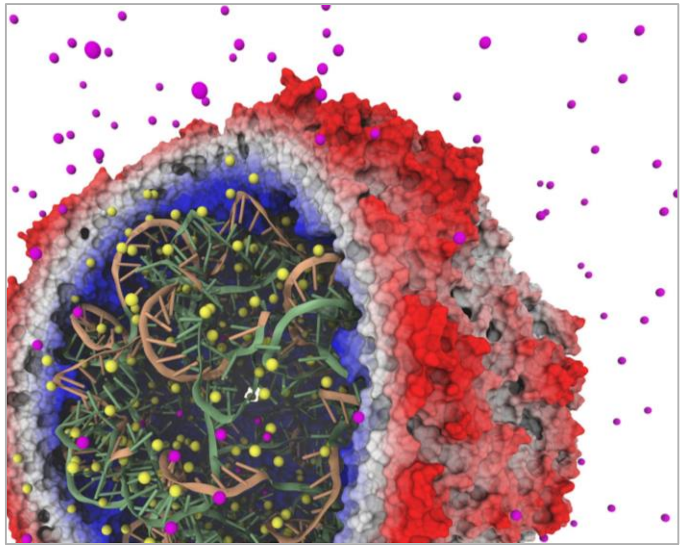
- Bonded forces are already upstreamed and available in Gromacs 2019. Upstreaming of all other components in progress.

PERFORMANCE RESULTS

BENCHMARKS

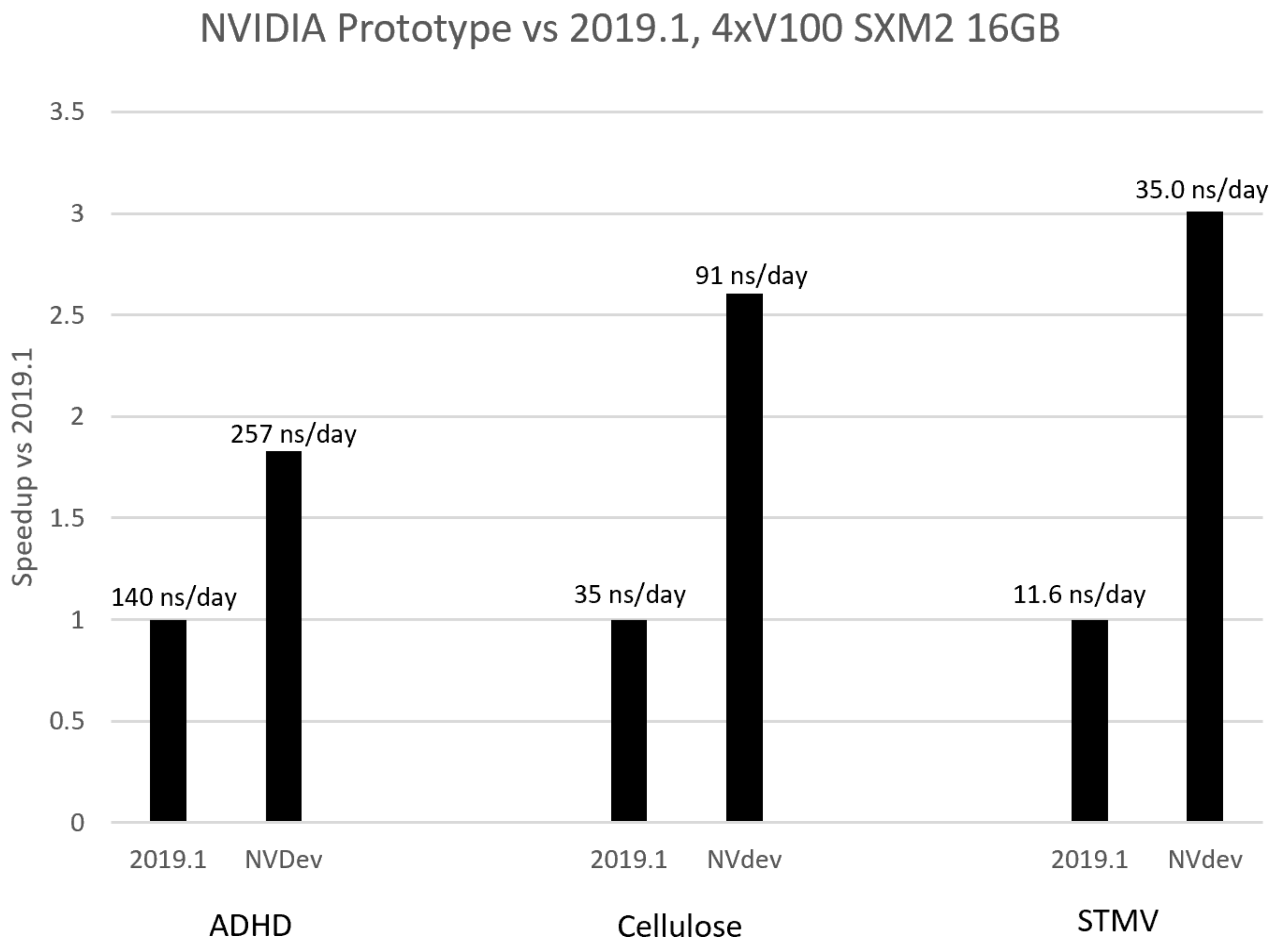

ADH Dodec
~100K atoms


Cellulose
~400K atoms

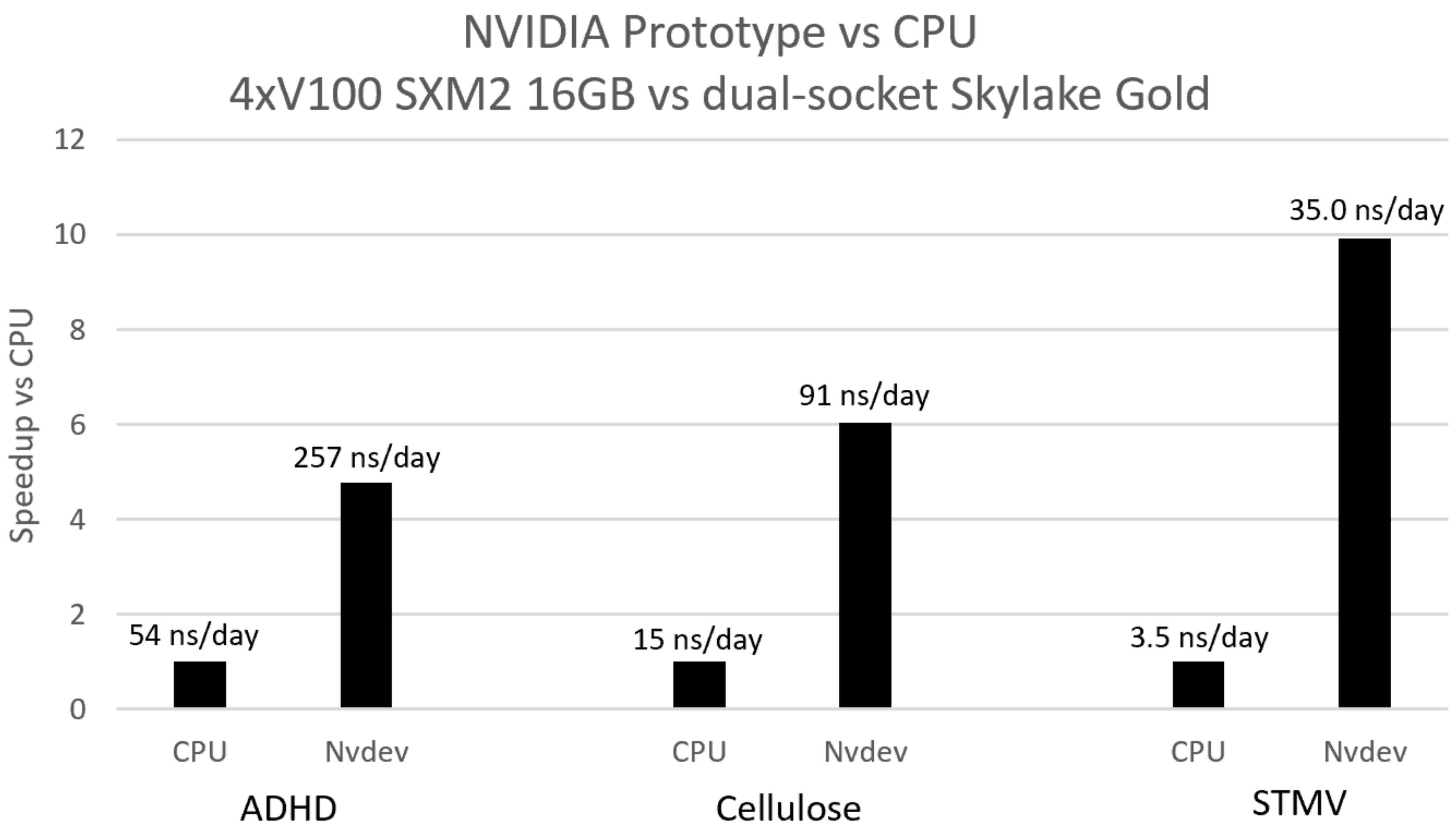

STMV
~1M atoms

- Performance results are dependent on system size. We strive to aim our benchmarking and optimization to cover the range of typical sizes in use. We welcome any feedback on further cases to include.

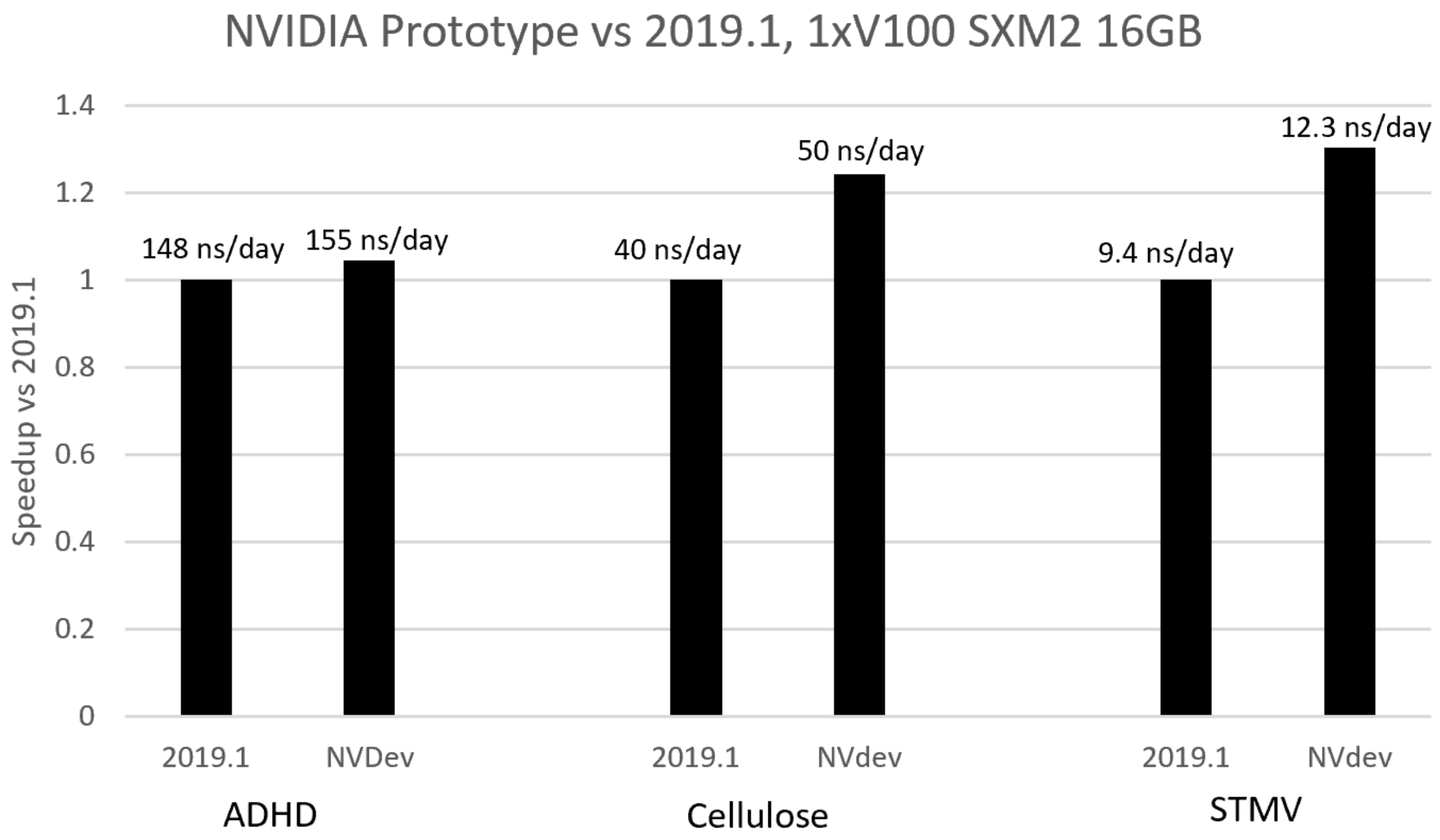
MULTI-GPU: PROTOTYPE VS GMX 2019.1



PROTOTYPE ON GPU VS 2019.1 ON CPU



SINGLE-GPU: PROTOTYPE VS GMX 2019.1



DEVELOPMENT DETAILS

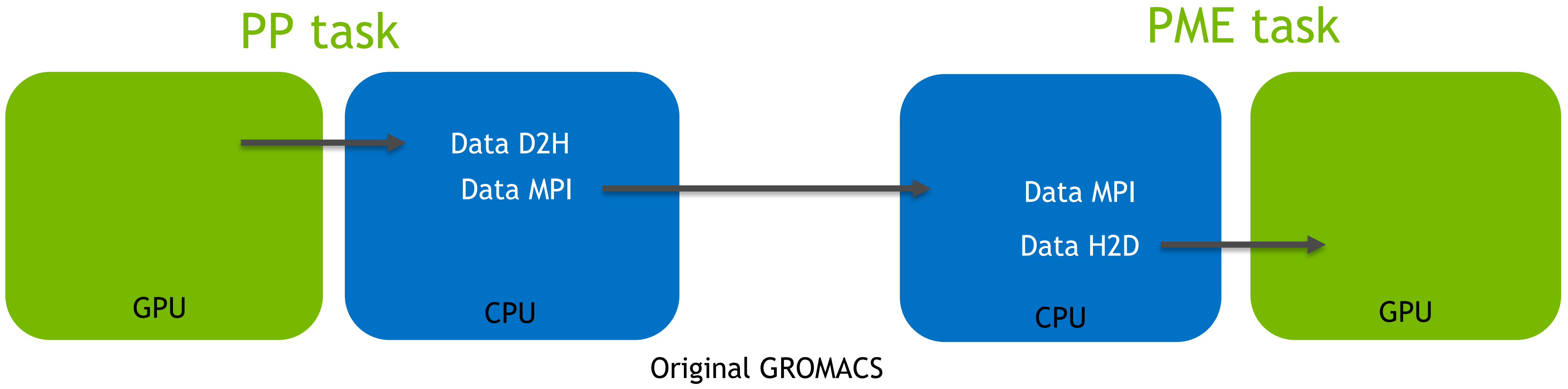
NVIDIA DEVELOPMENTS

- Reminder: Upstreaming of developments is in collaboration with core Gromacs developers.
- GPU Bonded: 8 new kernels corresponding to bonded force types
  - already integrated in Gromacs 2019
- GPU Buffer Ops: transformations between different data formats used in gromacs, and force reduction operations. 2 new kernels and restructuring.
  - Several patches to gromacs master branch in progress.

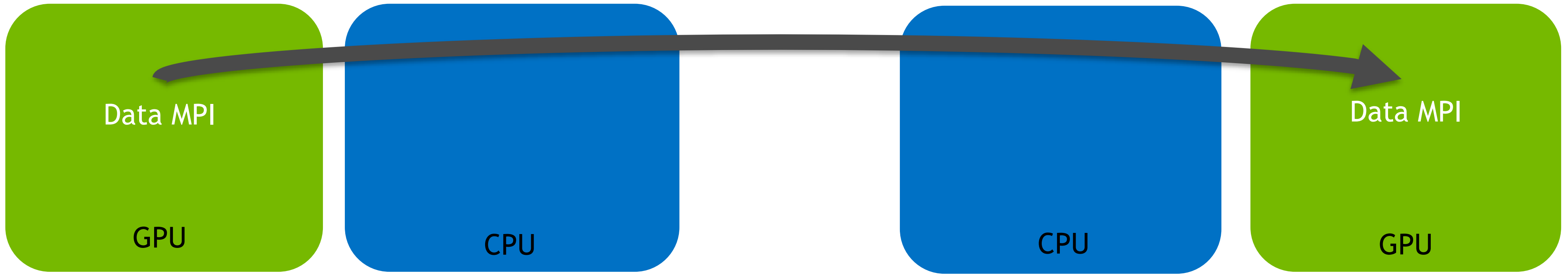
NVIDIA DEVELOPMENTS

- GPU Update and Constraints
  - 11 new kernels related to the “update”, “lincs” and “settle” operations to update and constrain atom positions from forces.
- Device MPI: PME/PP Gather and Scatter
  - Use of CUDA-aware MPI to exchange data directly between GPUs
  - More details coming up
- Device MPI: PP halo exchanges
  - New functionality to pack device-buffers and exchange directly between GPUs using CUDA-aware MPI
  - More details coming up
- Patches to master branch in progress for all the above

PP TO PME COMMUNICATION

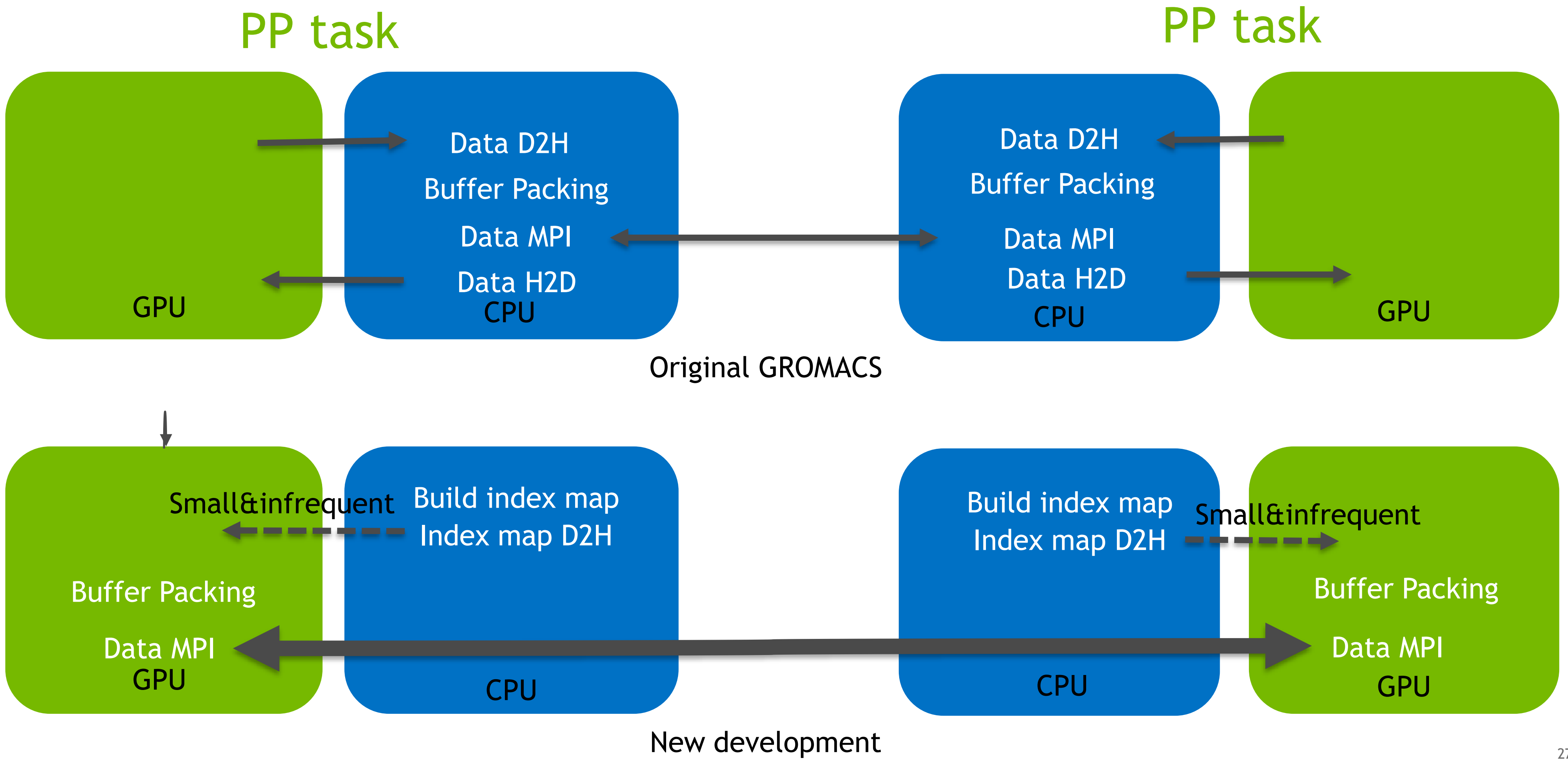


Original GROMACS



New development

PP TO PP HALO EXCHANGE COMMUNICATION



NEXT STEPS

- As described, integrate new developments into master branch
  - Such that they become available for GMX 2020 Beta release in Autumn 2019
- Further developments
  - Small case optimization:
    - Performance benefits currently more profound for larger cases.
    - Smaller cases are more sensitive to overheads associated with short GPU activities (e.g. kernel launch latency).
    - We can leverage new CUDA features such as **CUDA Graphs** to improve.
    - Also other improvements such as fusing kernels.
  - PME decomposition: enablement of multi-GPU for PME could improve load balance, and also potentially allow scaling to higher numbers of GPUs.

ATTACKING SMALL PROBLEM SIZES WITH CUDA GRAPHS

GETTING STARTED WITH CUDA GRAPHS

By way of simple example

Pattern occurring in real apps (including Gromacs)

- Loop over timesteps/iterations

…

shortKernel1

shortKernel2

…

shortKernelN

Section of timestep involving execution of multiple short kernels

…

GETTING STARTED WITH CUDA GRAPHS

By way of simple example

```
#define N 500000 // tuned such that kernel takes a few microseconds

__global__ void shortKernel(float * out_d, float * in_d){

    int idx = blockIdx.x * blockDim.x + threadIdx.x;

    if(idx < N){
        out_d[idx] = 1.23 * in_d[idx];
    }

    return;
}
```

- Simple kernel devised to represent a real short-lasting kernel
- Can use profiler to measure execution time: 2.9μs on V100 (CUDA 10.1, 512 threads per block)
- Can call repeatedly to mimic patterns found in real apps

GETTING STARTED WITH CUDA GRAPHS

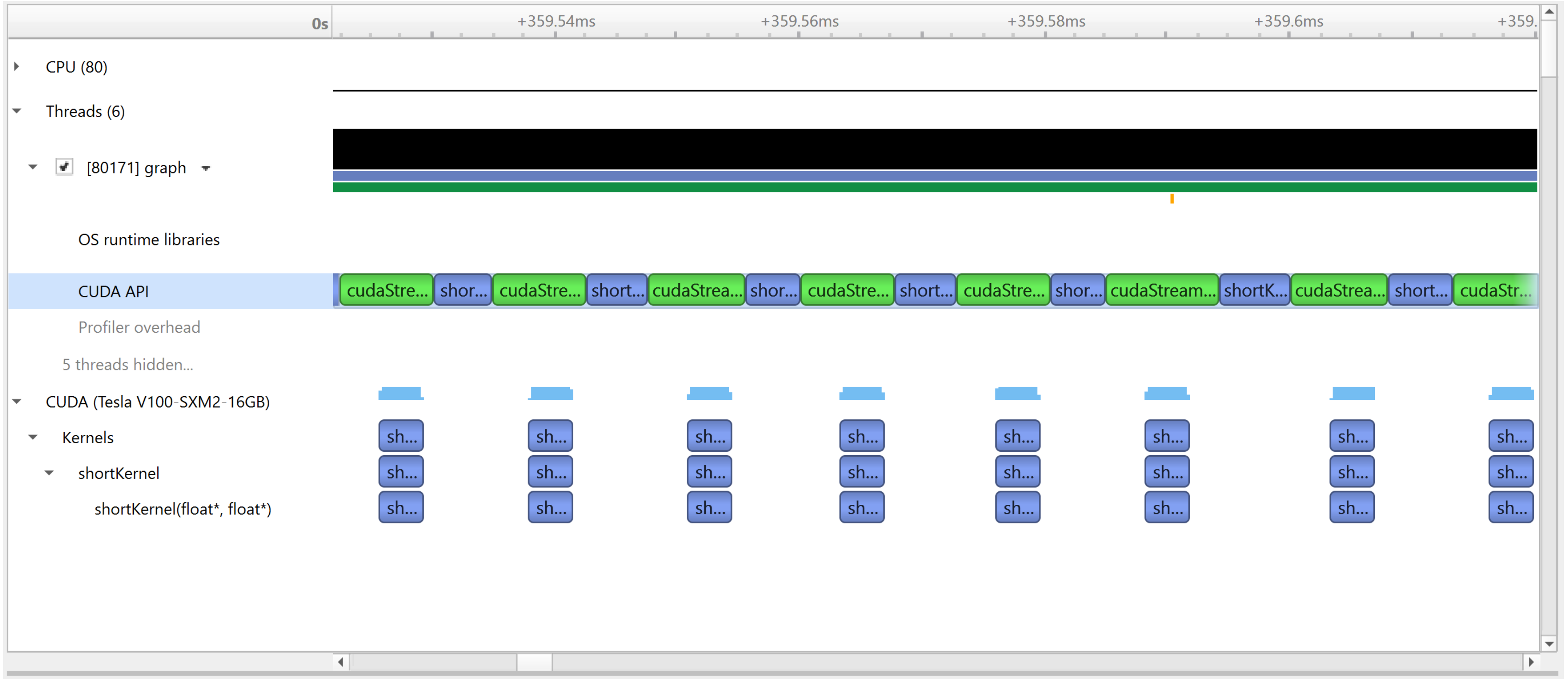
By way of simple example

```
#define NSTEP 1000
#define NKRNL 20

// start wallclock timer
for(int step=0; step<NSTEP; step++){
   for(int krnl=0; krnl<NKRNL; krnl++){
     shortKernel<<<blocks, threads, 0, stream>>>(out_d, in_d);
     cudaStreamSynchronize(stream);
   }
}
//end wallclock timer
```

- Call kernel 20 times, each of 1000 iterations.
- Time taken per kernel inc overheads: 9.6 μs (vs 2.9μs execution time).
- But note that with above code, each kernel is not launched until previous completes
    - No overlap of launch overhead with computation

GETTING STARTED WITH CUDA GRAPHS



- Launch overheads are fully exposed
- NB: profiler adds some overhead on this timescale

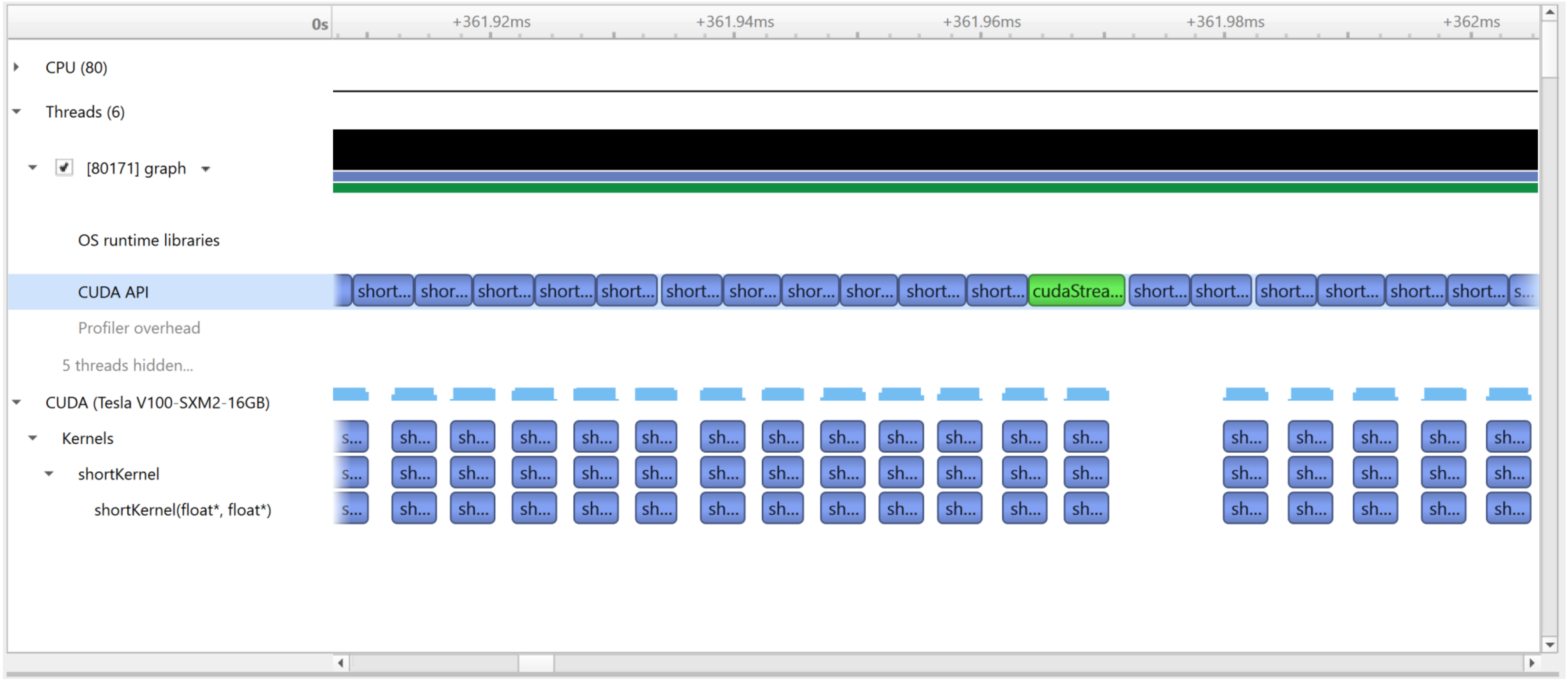
GETTING STARTED WITH CUDA GRAPHS

By way of simple example

```
// start wallclock timer
for(int step=0; step<NSTEP; step++){
   for(int krnl=0; krnl<NKRNL; krnl++){
     shortKernel<<<blocks, threads, 0, stream>>>(out_d, in_d);
   }
   cudaStreamSynchronize(stream);
}
//end wallclock timer
```

- Move sync out of inner loop: allow overlap of launch overhead with computation.
- Time taken per kernel inc overheads: 3.8 μs (vs 2.9μs execution time).
- Better, but still overheads associated with multiple launches.

GETTING STARTED WITH CUDA GRAPHS



- Launch overheads are partially hidden, but overheads remain. Instead, can use Graphs to launch all the kernels (within an iteration) in a single operation.
- NB: profiler adds some overhead on this timescale

GETTING STARTED WITH CUDA GRAPHS

By way of simple example

```
bool graphCreated=false;
cudaGraph_t graph;
cudaGraphExec_t instance;
for(int step=0; step<NSTEP; step++){
   if(!graphCreated){
      cudaStreamBeginCapture(stream,cudaStreamCaptureModeGlobal);
      for(int krnl=0; krnl<NKRNL; krnl++){
         shortKernel<<<blocks, threads, 0, stream>>>(out_d, in_d);
      }
      cudaStreamEndCapture(stream,&graph);
      cudaGraphInstantiate(&instance,graph,NULL,NULL,0);
      graphCreated=true;
   }
   cudaGraphLaunch(instance, stream);
   cudaStreamSynchronize(stream);
}
```

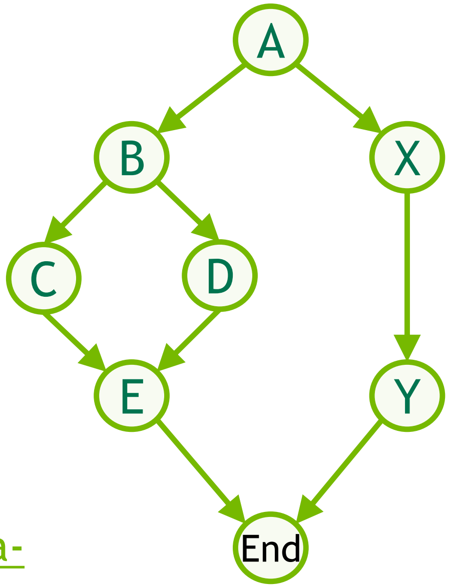
Stream capture into graph, only on first iteration

Launch graph in a single operation

- Time taken per kernel inc overheads: 3.4 μs (vs 2.9μs execution time).
- Future work in CUDA will aim to further improve overheads.

CUDA GRAPHS

- In this very simple case, most of the overhead was already being hidden - use of CUDA Graphs able to further decrease the overhead.
- More complex cases provide more opportunities for savings.
  - Multiple interacting streams with different types of GPU operations.
  - Graphs may span multiple GPUs
- Can define using stream capture or directly using API.
- S9240: CUDA – New Features and Beyond, Stephen Jones (NVIDIA)
- Programming Guide:
  - https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#cuda-graphs
- CUDA sample: `samples/0_Simple/simpleCudaGraphs`



SUMMARY

- Modern GPUs are so fast in performing Gromacs force calculations that the other parts of the simulation timestep are becoming a bottleneck.
- We showed results from accelerating the other computational parts and enabling peer-to-peer communication directly between GPUs.
- Our prototype shows large performance increases over the released version of Gromacs.
- We are now working with the core Gromacs developers to integrate these into the main branch, and perform further improvements.
- For small cases, approaches are required which minimize overheads associated with short operations. We gave a demonstration of how CUDA Graphs can be used for this sort of problem.

NVIDIA®