center for
excellence in parallel
programming

CUDA

Georges-Emmanuel Moulard
Paul Karlshöfer

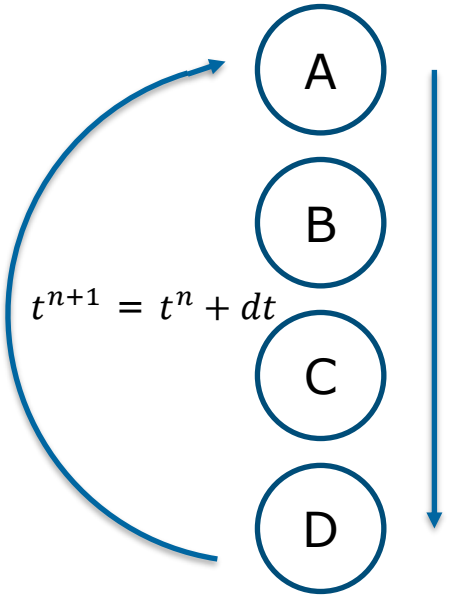
Bull
atos technologies

CUDA Graphs

05/08/2020

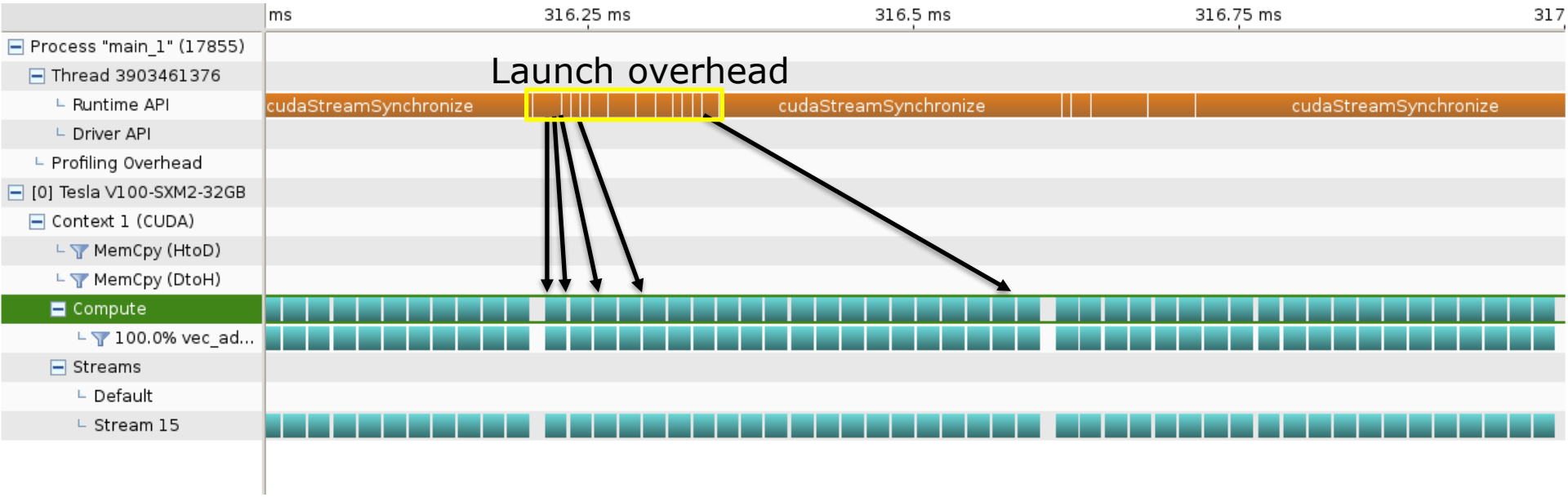
CUDA Graphs – An example

- Time depended Systems of differential equations have to be discretized in time. This leads to compute loops, controlled by some sort of timestepping schema.
- Within a loop, the operations are often similar. (special discretization)
- The operations can be formulated as a number of compute kernels (A,B,C,D)
- Each kernel is relatively small



CUDA Graphs – An example

- Naive approach: Launch each kernel separately in the compute loop
- Problem: each kernel launch has overhead!
  - Even if pushed in a stream at once



CUDA Graphs

- Launch all kernels for each timestep iteration in one operation.

- Idea:
  - Record all kernel launches to a stream (without executing them)
  - Build executable instance (a CUDA `graphExec_t`)
  - Launch the instance once to queue all operations in the stream

- requires at least CUDA 10

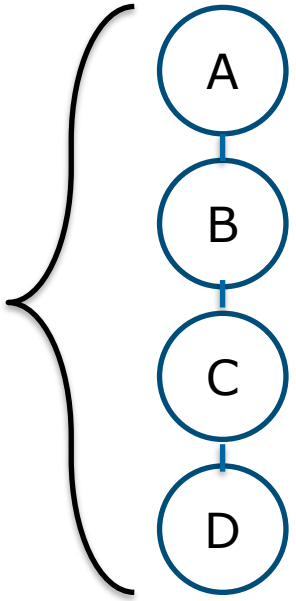
CUDA Graphs

```
cudaGraph_t graph;
cudaGraphExec_t instance;

cudaStreamBeginCapture(stream);
//launch kernels
cudaStreamEndCapture(stream, &graph);

cudaGraphInstantiate(&instance, graph, NULL, NULL, 0);

while(t < t_end){                    //compute loop
    cudaGraphLaunch(instance, stream);
    cudaStreamSynchronize(stream);
}
```

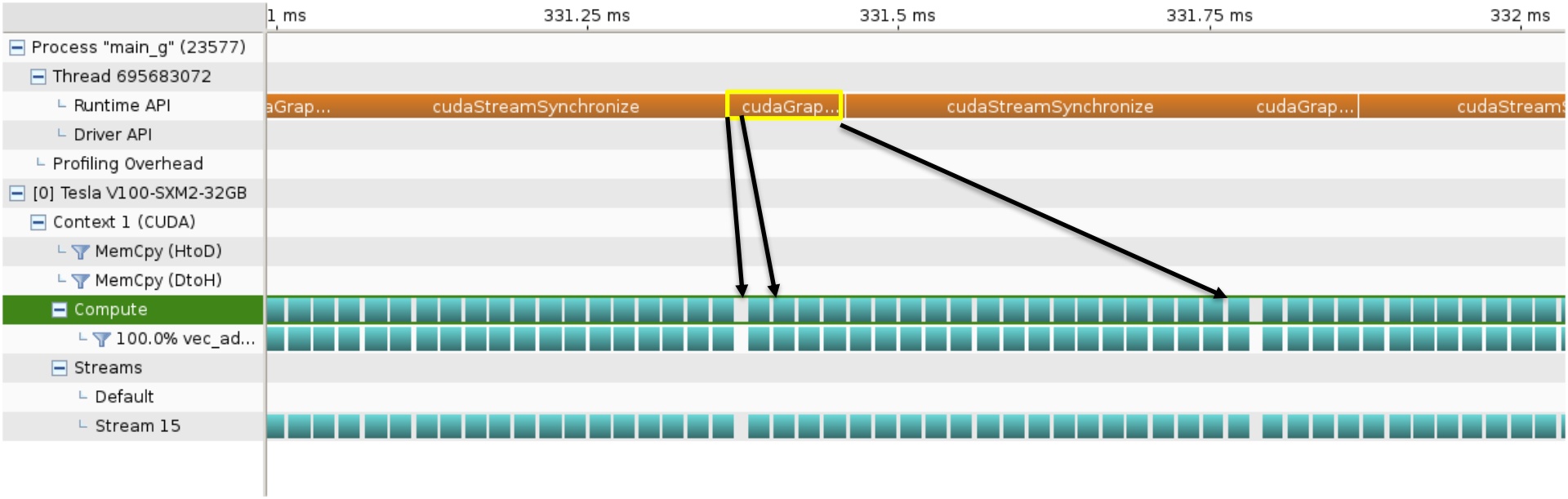


TP: CUDA Graphs

- TP_CUDA_C/GRAPHS/1/

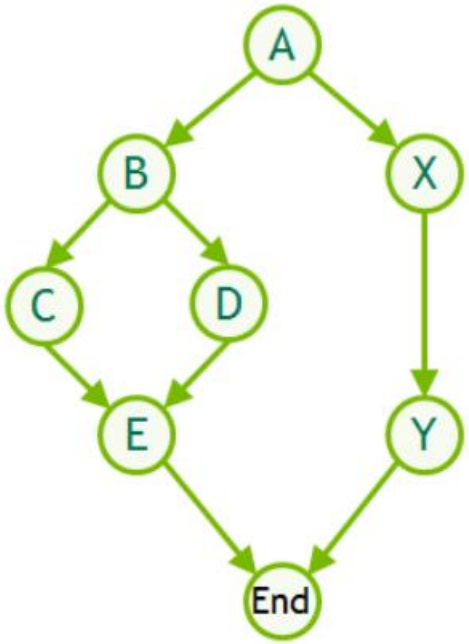
CUDA Graphs

- With a CUDA Graph we cluster all kernel launches



CUDA Graphs

- CUDA Graphs improve performance, where many small kernels are repeatedly launched in the same order
- Handle complex program flow
- Can be formally described via the `simpleCUDAGreaphs` API
  - or being captured as shown above
- Use Events to control execution flow amongst streams



Copyright

Thanks

For more information please contact:
Paul Karlshöfer
paul.karlshoefer@atos.net

29-10-2018

Bull
atos technologies