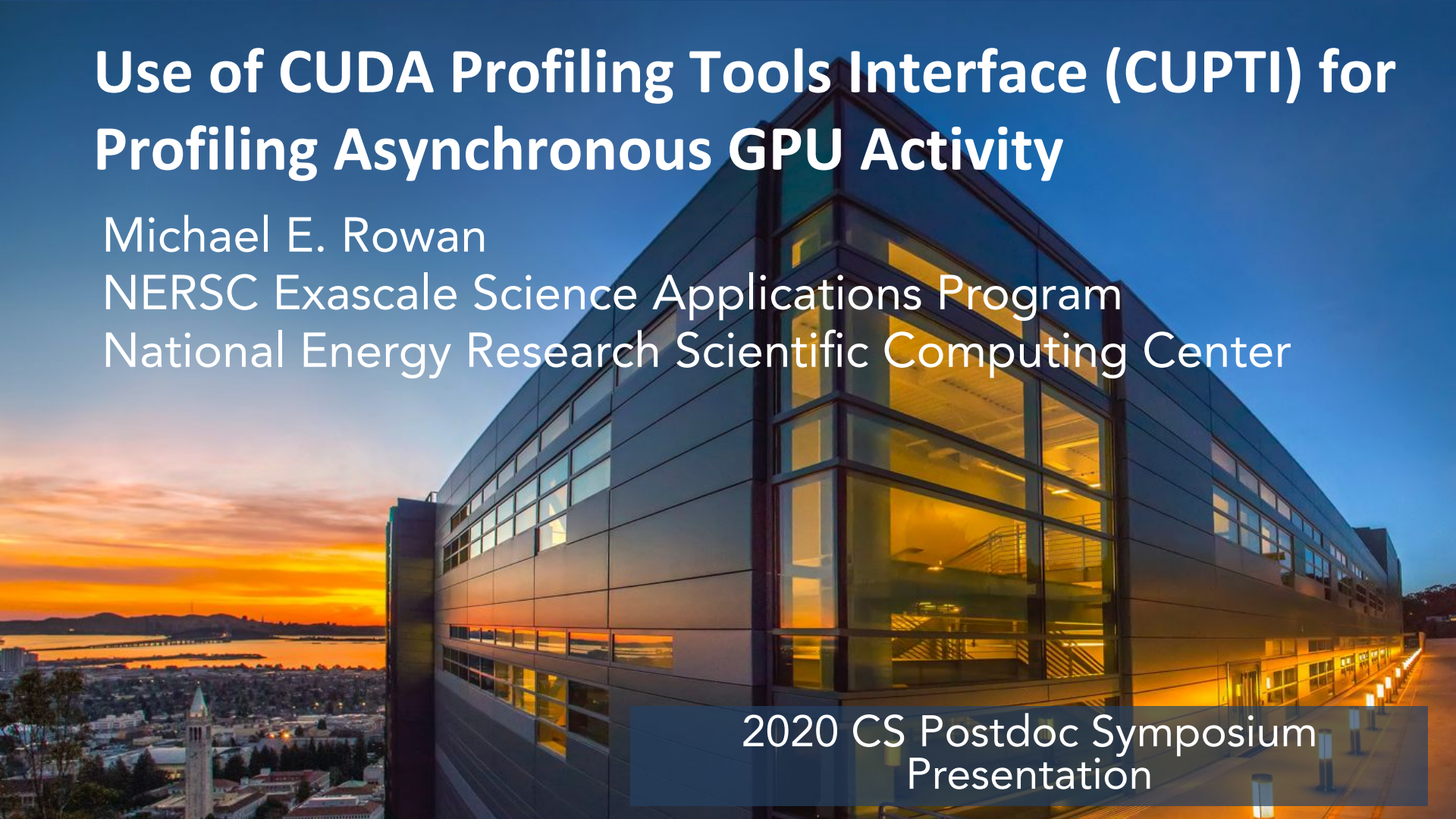
Use of CUDA Profiling Tools Interface (CUPTI) for Profiling Asynchronous GPU Activity

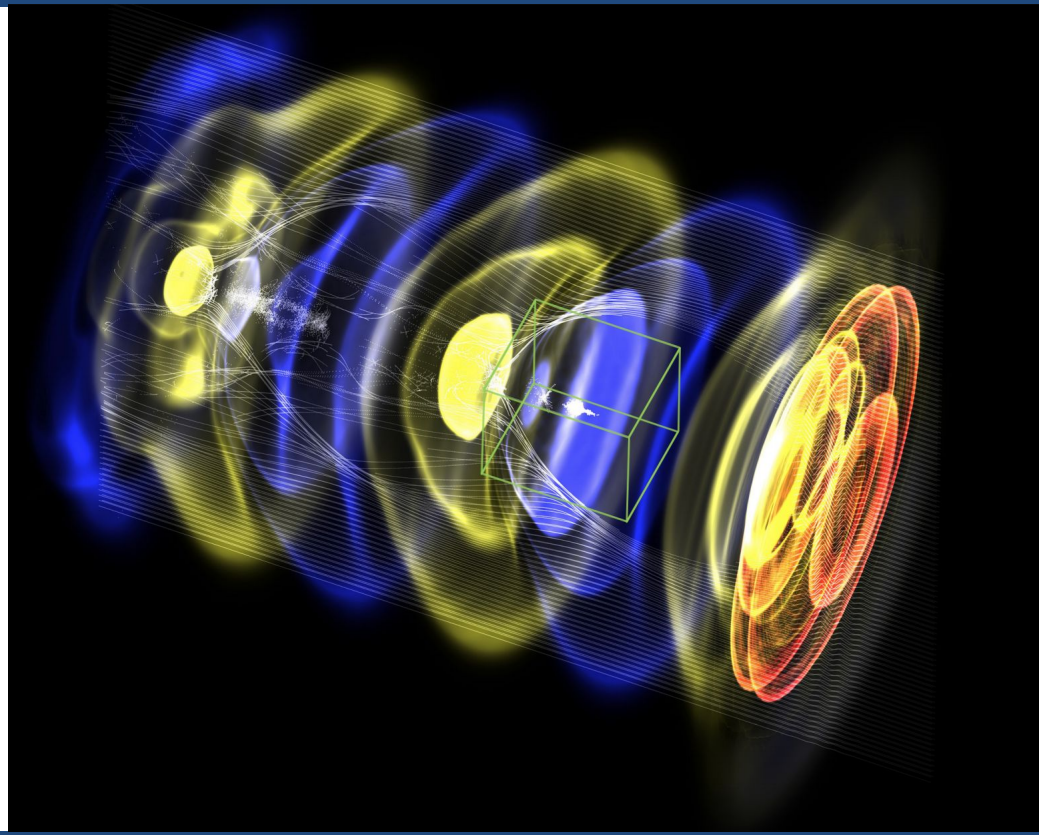
Michael E. Rowan
NERSC Exascale Science Applications Program
National Energy Research Scientific Computing Center



2020 CS Postdoc Symposium Presentation

Real-time measurement of kernel execution time is needed for correct load balancing in GPU-accelerated codes

- On-the-fly measurement not possible with standard profiling tools (NVProf, Nsight)
- Developed a method (CUPTI Callback timing) for real-time measurement of kernel time
- Impact:
  - Provides accurate kernel timing **on-the-fly**
  - Enables correct **load balancing** in WarpX



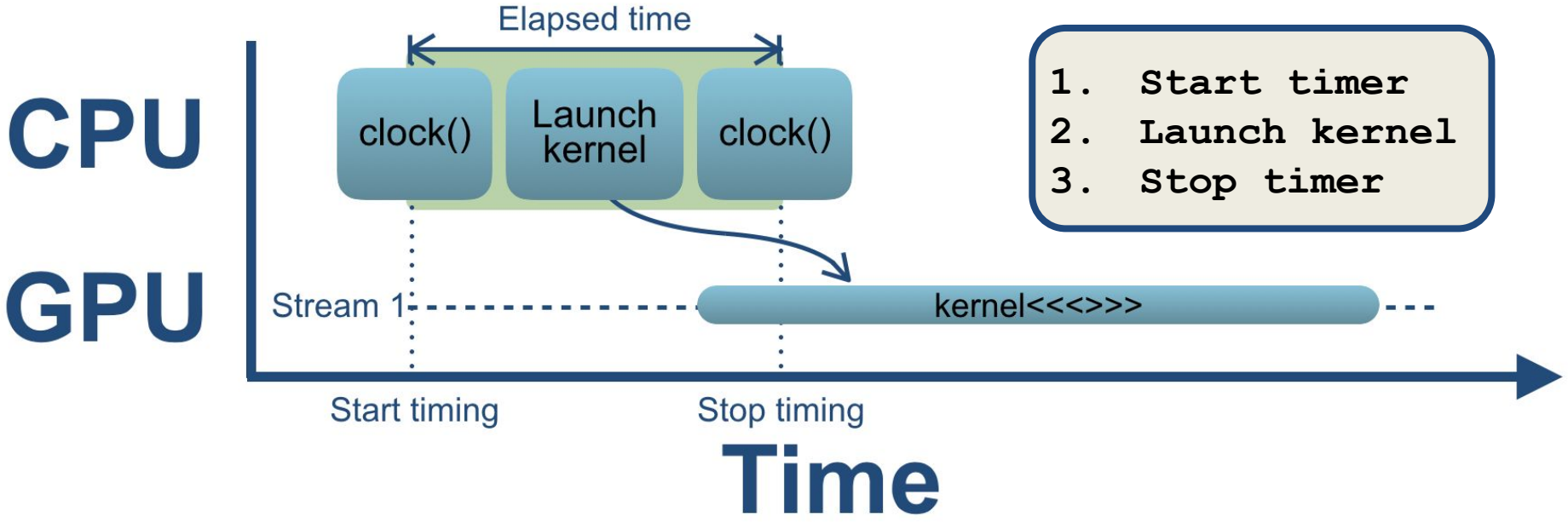
Cannot measure execution time on GPU the same way as on CPU!

- What happens if we try to measure kernel execution time naively?

```
1.  Start timer
2.  Launch kernel
3.  Stop timer
```

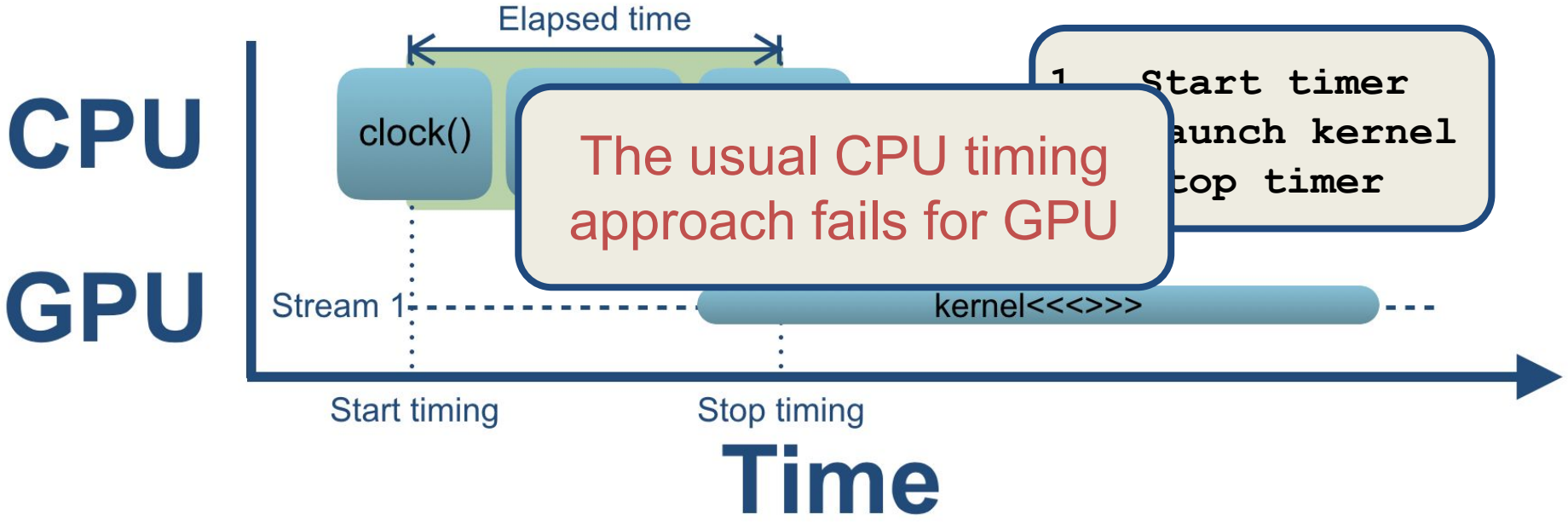
Cannot measure execution time on GPU the same way as on CPU!

- What happens if we try to measure kernel execution time naively?



Cannot measure execution time on GPU the same way as on CPU!

- What happens if we try to measure kernel execution time naively?



GPU work is in general asynchronous

- GPU operations are asynchronous with respect to:
  - Streams (series of operations which execute in issue order)
    - Operations across streams may be interleaved
    - While operations within a stream execute in-order, there is no relationship between issue order execution order for operations in different streams
  - Host
    - Kernel execution, e.g., is by default asynchronous with host

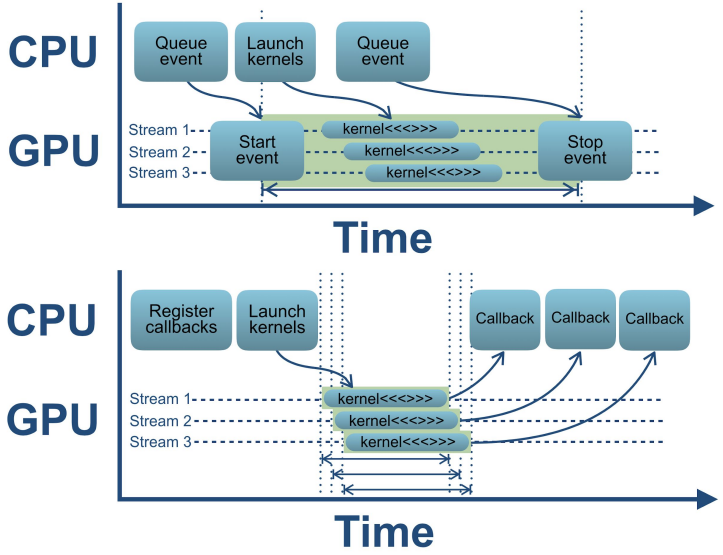
```
kernel<<<...>>>(...)
cpuWork(...)
```

} May overlap, as kernel launch is non-blocking

Three possible on-the-fly timing strategies

1. Count GPU clock cycles
   - Requires additional device-to-host transfers
   - Implementation may be invasive
2. CUDA Events
   - Can give ambiguous results
3. CUDA Profiling Tools Interface (CUPTI)
   - Buffer requests and delivery of timing information handled by CUPTI
   - Gives unambiguous kernel timings

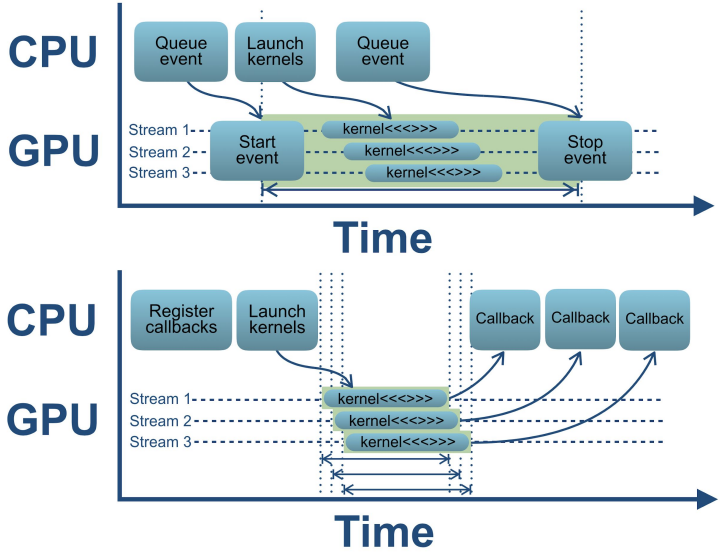
```
1.  Start GPU timer
2.  Do GPU kernel work
3.  Stop GPU timer
4.  Send elapsed time to host
```



Three possible on-the-fly timing strategies

1. Count GPU clock cycles
   - Requires additional device-to-host transfers
   - Implementation may be invasive
2. CUDA Events
   - Can give ambiguous results
3. CUDA Profiling Tools Interface (CUPTI)
   - Buffer requests and delivery of timing information handled by CUPTI
   - Gives unambiguous kernel timings

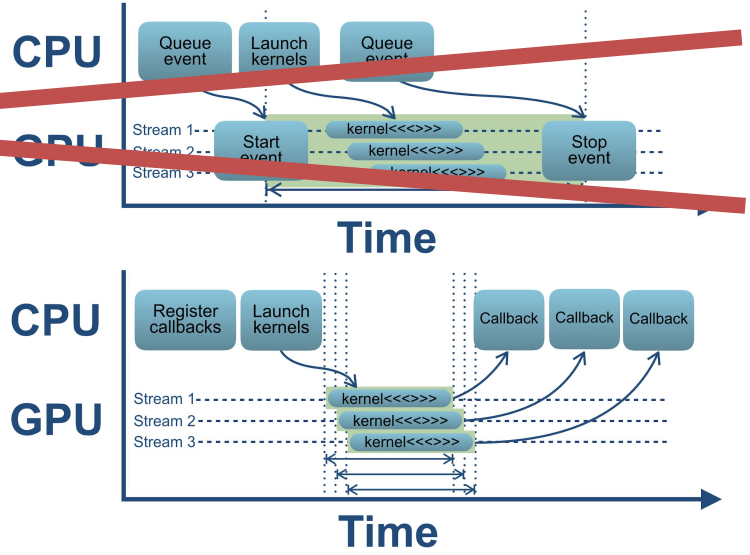
```
1.   Start GPU timer
2.   Do GPU kernel work
3.   Stop GPU timer
4.   Send elapsed time to host
```



Three possible on-the-fly timing strategies

1. Count GPU clock cycles
   - Requires additional device-to-host transfers
   - Implementation may be invasive
2. CUDA Events
   - Can give ambiguous results
3. CUDA Profiling Tools Interface (CUPTI)
   - Buffer requests and delivery of timing information handled by CUPTI
   - Gives unambiguous kernel timings

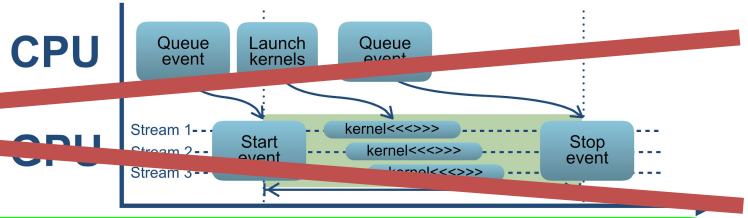
```
1.  Start GPU timer
2.  Do GPU kernel work
3.  Stop GPU timer
4.  Send elapsed time to host
```



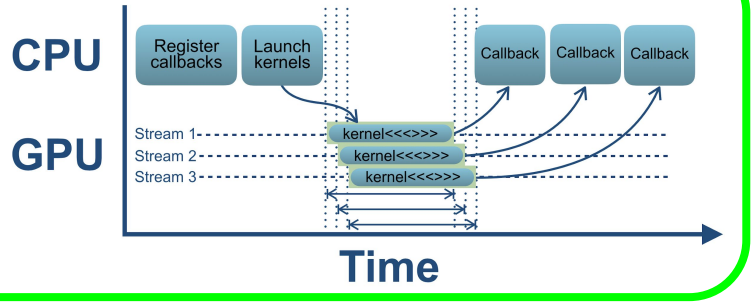
Three possible on-the-fly timing strategies

1. Count GPU clock cycles
   - Requires additional device to host transfers
   - Implementation may be invasive
2. CUDA Events
   - Can give ambiguous results

```
1.  Start GPU timer
2.  Do GPU kernel work
3.  Stop GPU timer
4.  Send elapsed time to host
```



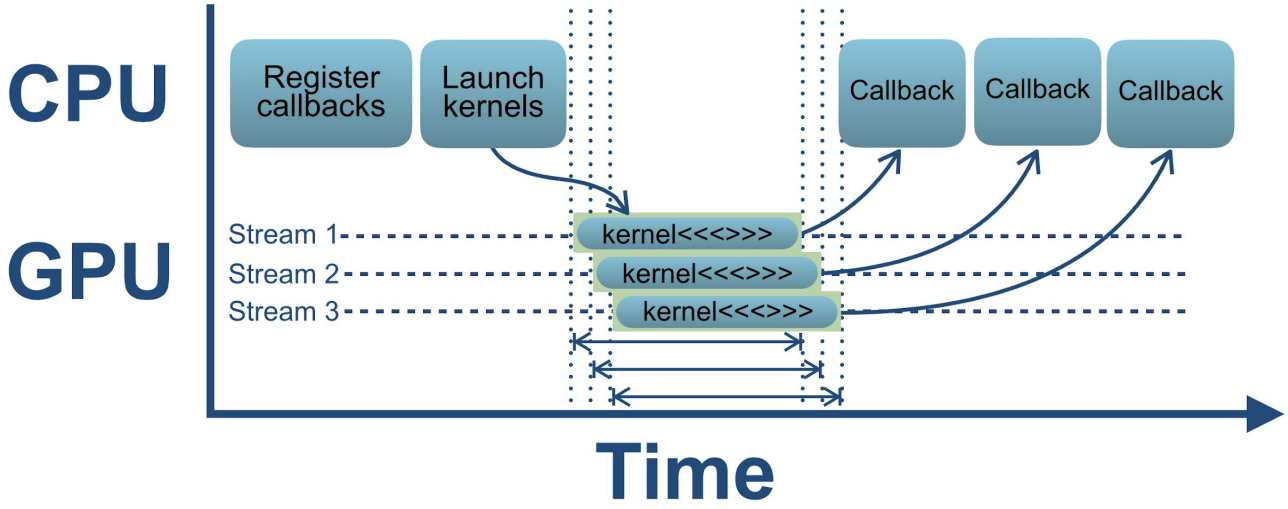
3. CUDA Profiling Tools Interface (CUPTI)
   - Buffer requests and delivery of timing information handled by CUPTI
   - Gives unambiguous kernel timings



Adopted solution: CUDA Profiling Tools Interface (CUPTI)

- Register callback functions to manage buffer request/delivery of ‘activity records’
- Callbacks triggered by GPU activity
- Access returned records

holds information about GPU or operations on GPU; different kinds for kernels, memory transfers, etc.



Timing with CUPTI Callback functions consists of just a few steps:

- Initialize trace:
  - Enable collection of kernel activity records
  - Register callback functions

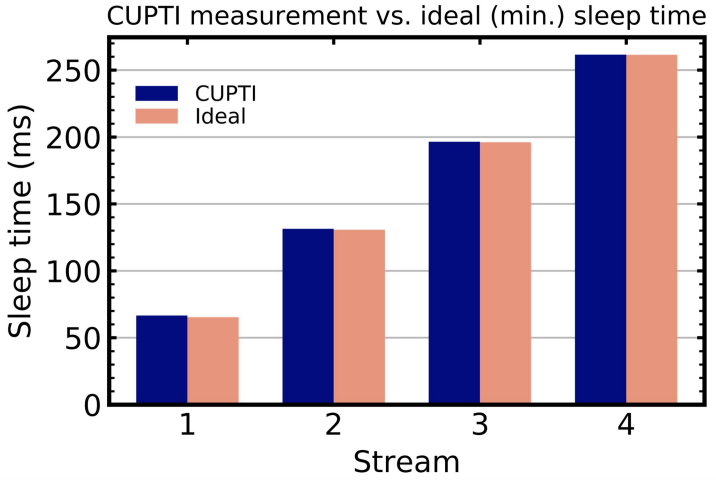
```
cuptiActivityEnable(CUPTI_ACTIVITY_KIND_CONCURRENT_KERNEL);
cuptiActivityRegisterCallbacks(bfrRequest, bfrCompleted);
```

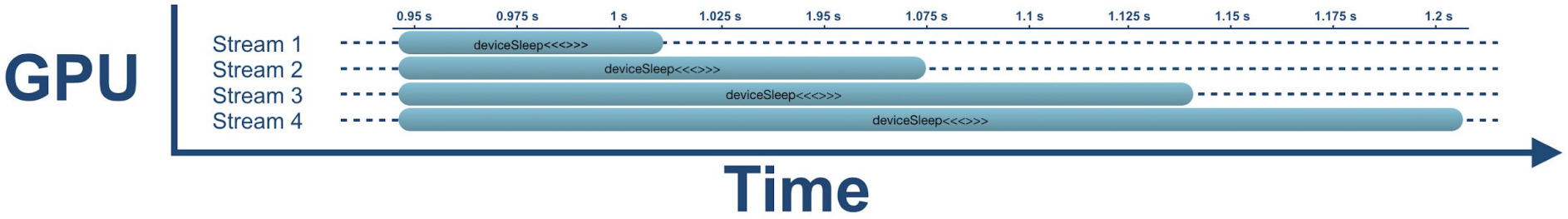
- Trigger callback functions; schematically, they look like this:

```
void CUPTI API bfrRequest (uint8_t **bfr, ...)
{
     // Signal to CUPTI client that an empty buffer is needed by CUPTI
}
void CUPTI API bfrCompleted (uint8_t *bfr, ...)
{
     // Return a buffer of completed activity records to CUPTI client
}
```

Results: implemented CUPTI-Callback timer in AMReX (Adaptive Mesh Refinement library), and tested with simple kernels

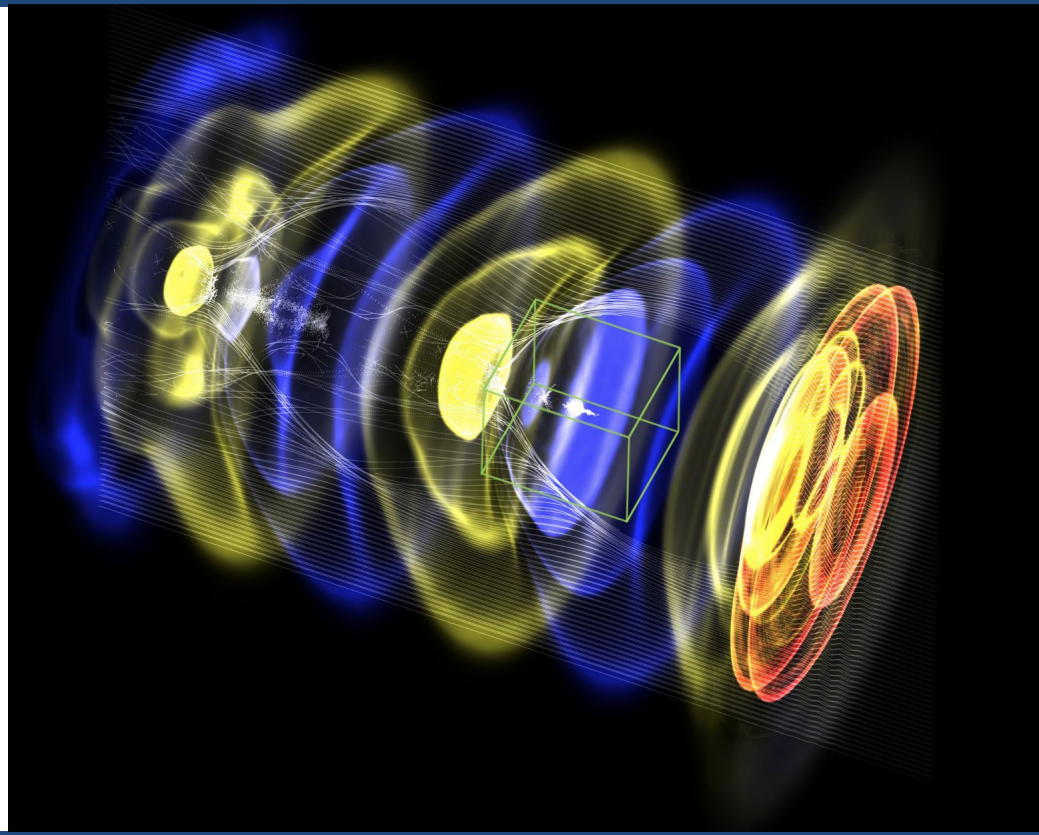
- We tested the CUPTI-based timer using a simple device sleep function
- With NVIDIA Volta V100 (peak clock frequency: 1.53 GHz), we launched sleep kernels on separate streams for multiples of 1, 2, 3, and 4×10^8 cycles (≈ 65 ms)





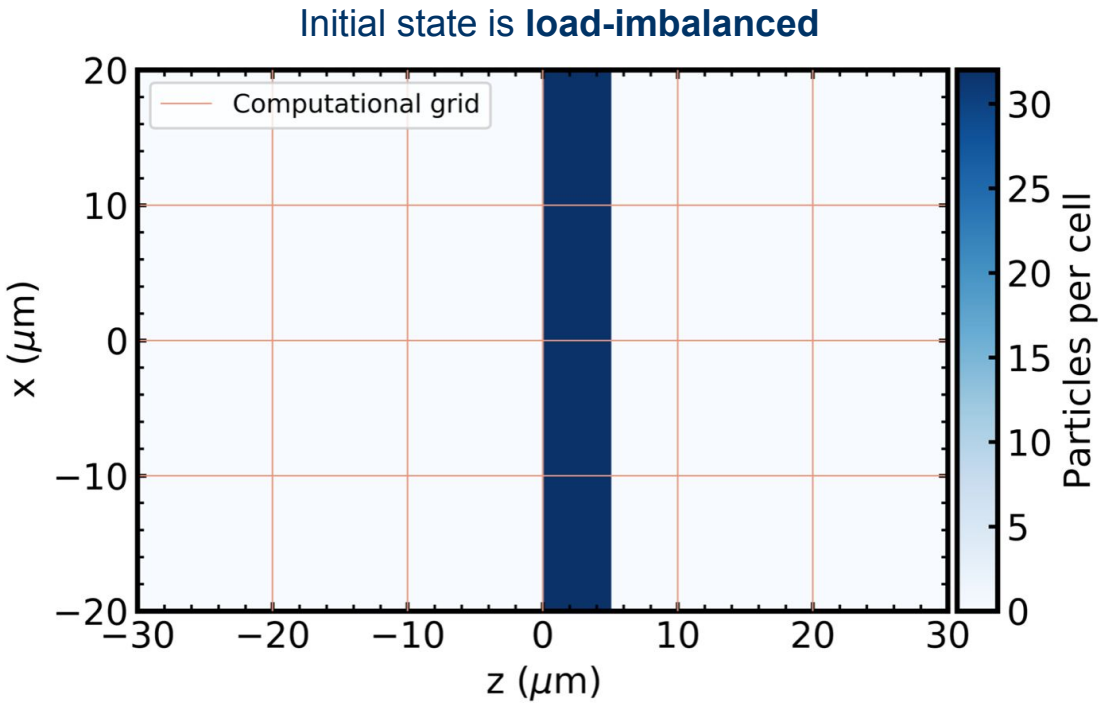
Tested the CUPTI Callback timer in a more realistic use case, the advanced electromagnetic particle-in-cell code WarpX

- WarpX: Advanced Electromagnetic Particle-in-Cell Code
  - Simulates laser wakefield acceleration with mesh refinement
  - Built on AMReX
- With the CUPTI Callback timer now implemented in AMReX, load balancing which properly accounts for GPU work is possible in WarpX



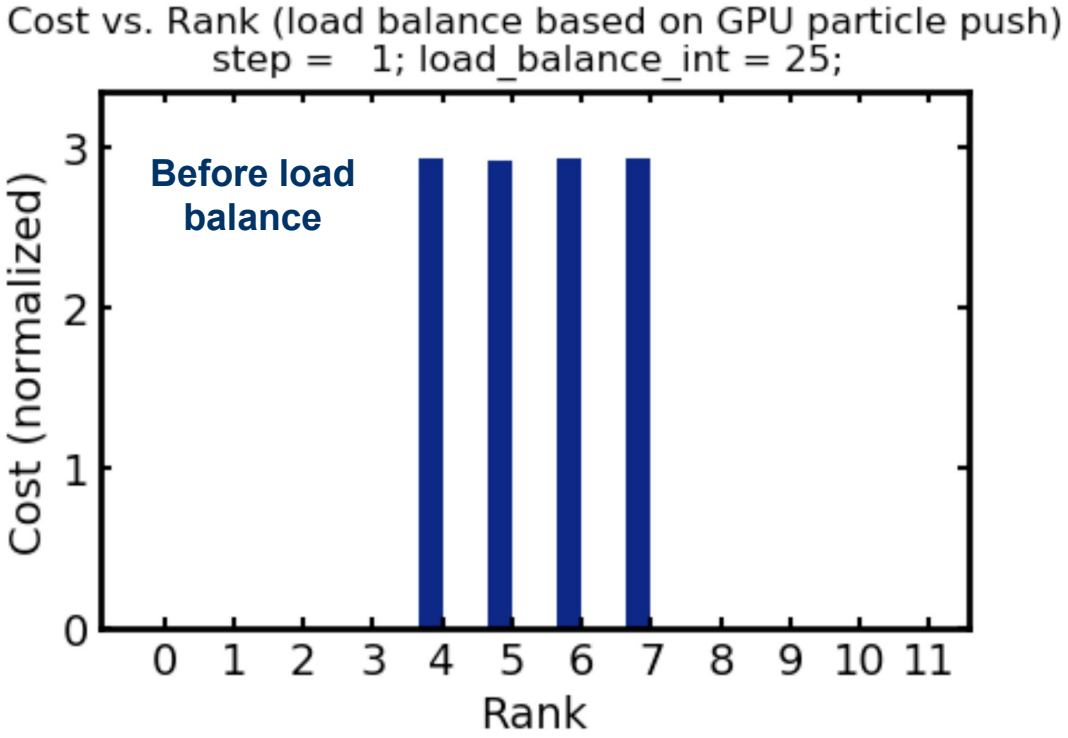
Used CUPTI Callback timing as input to load balancing modules for a WarpX test problem

- Plot to the right is the 2D average along *y* of particles per cell at $t = 0$
- Dark blue stripe is a region of high particle density
- White region has no particles
- Light red lines show domain decomposition
- Load-imbalanced problem, by construction



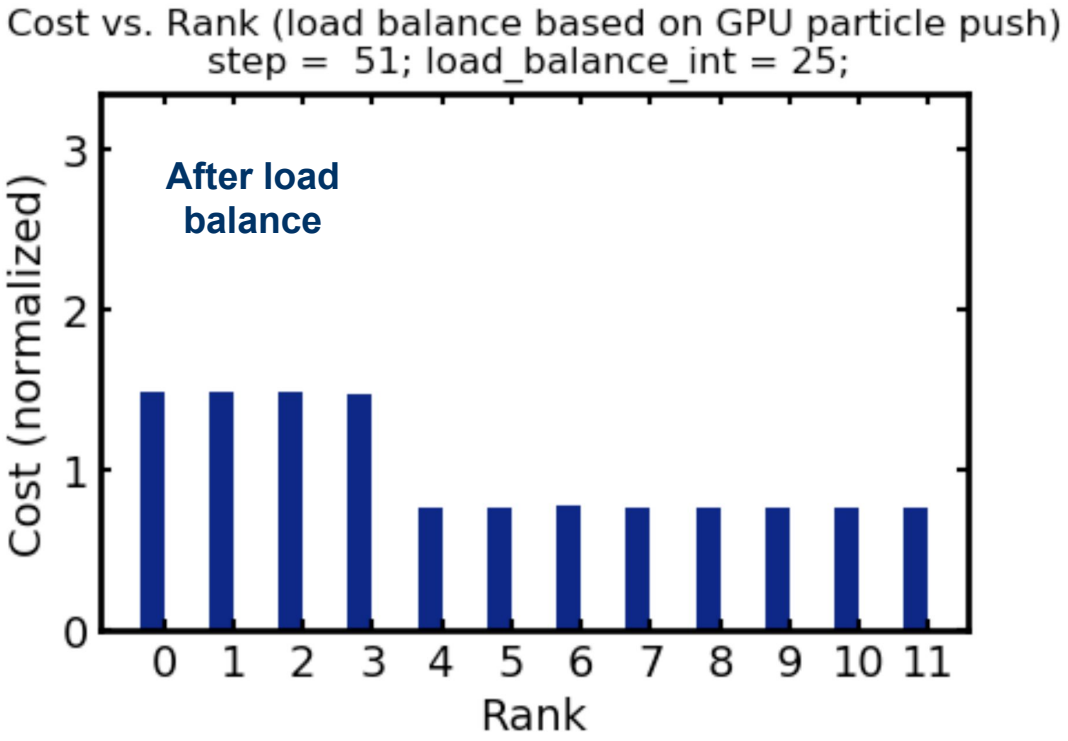
Used CUPTI Callback timing as input to load balancing modules for a WarpX test problem

- Plot to the right shows time evolution of ‘cost’ per GPU
- Load balance every 25 steps (for this case)
- Cost initially imbalanced (ranks, 4 – 7 do most of the work)
- Work is more evenly distributed after step 25



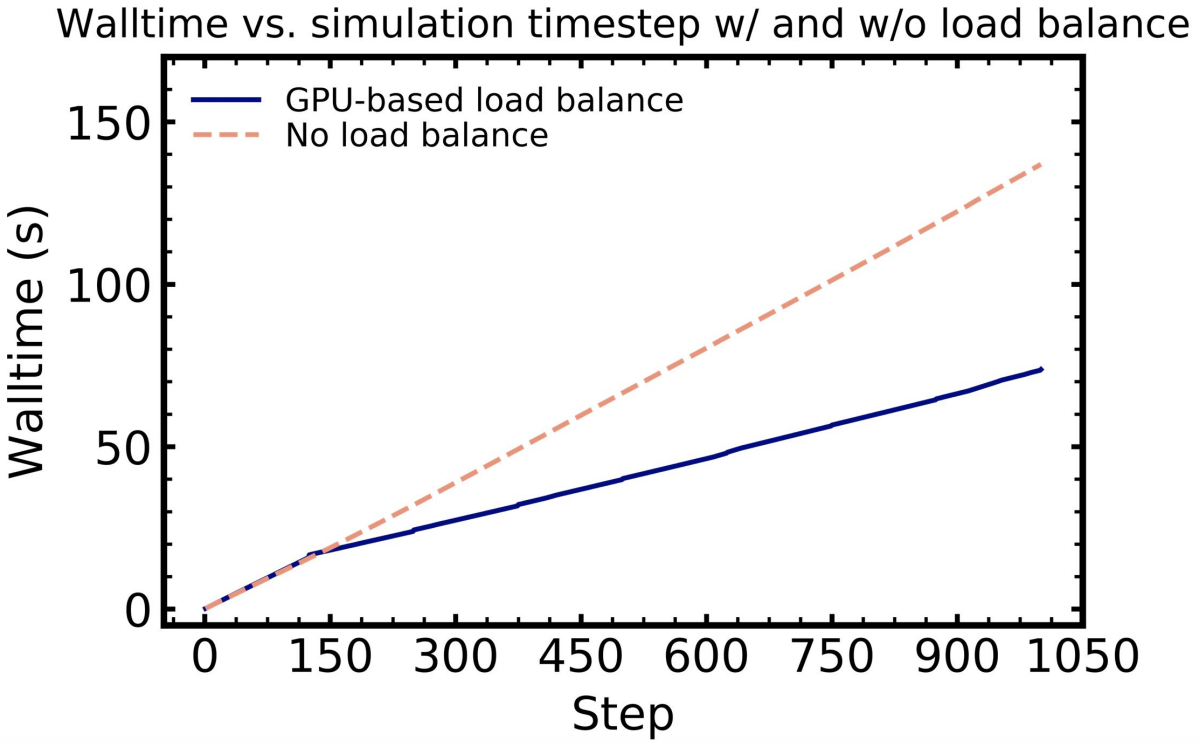
Used CUPTI Callback timing as input to load balancing modules for a WarpX test problem

- Plot to the right shows time evolution of ‘cost’ per GPU
- Load balance every 25 steps (for this case)
- Cost initially imbalanced (ranks, 4 – 7 do most of the work)
- Work is more evenly distributed after step 25



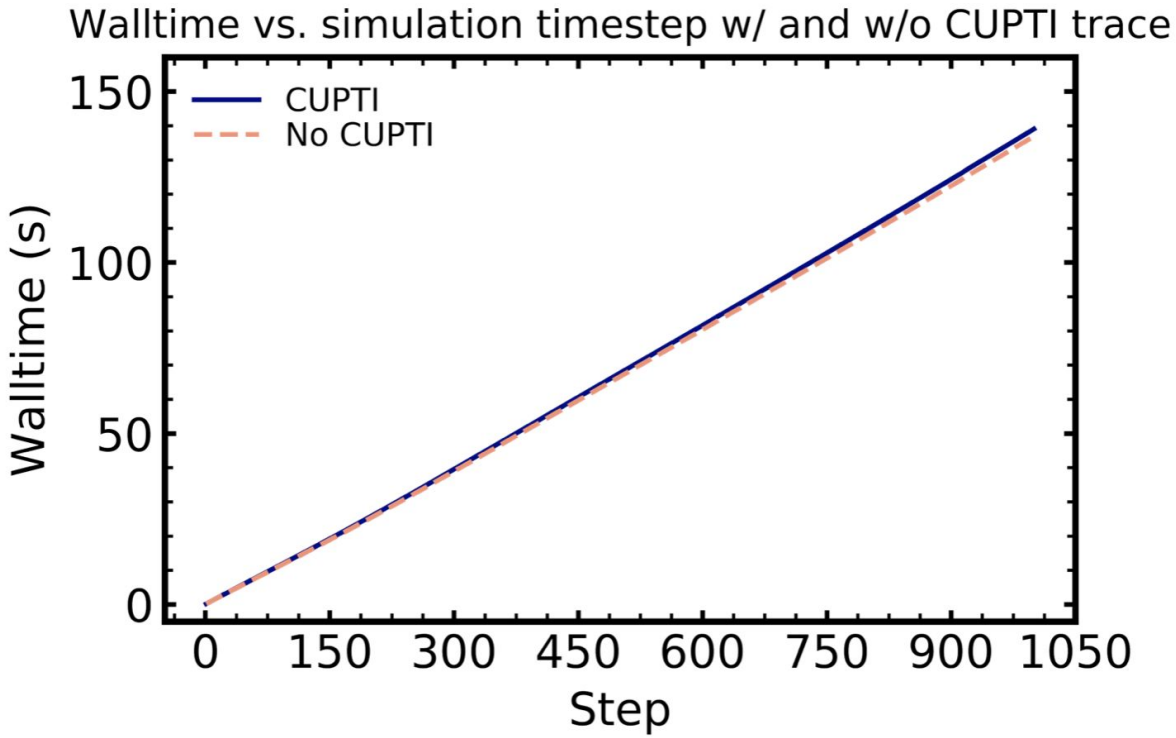
In this test case, get a speedup of ~2x

- Red curve shows case with no load balancing
- Blue curve shows case with load balance every 125 steps
- Prior to the first load balance (step=125), work is unevenly distributed over GPUs
- After load balance, work is distributed more evenly → performance improvement



Overhead incurred with CUPTI timers is small

- Comparison of walltime for WarpX simulation with CUPTI initialized (dark blue) and without CUPTI (light red)
- Agreement between red and blue demonstrates that there is only a small overhead when using CUPTI



Conclusion

- Developed a technique (CUPTI Callback timing) for real-time GPU kernel profiling
- Implemented in AMReX
- Impact:
  - Provides accurate kernel timing **in real time**
  - Enables correct **load balancing** in WarpX
- Future work:
  - Load balancing in ion acceleration problem