

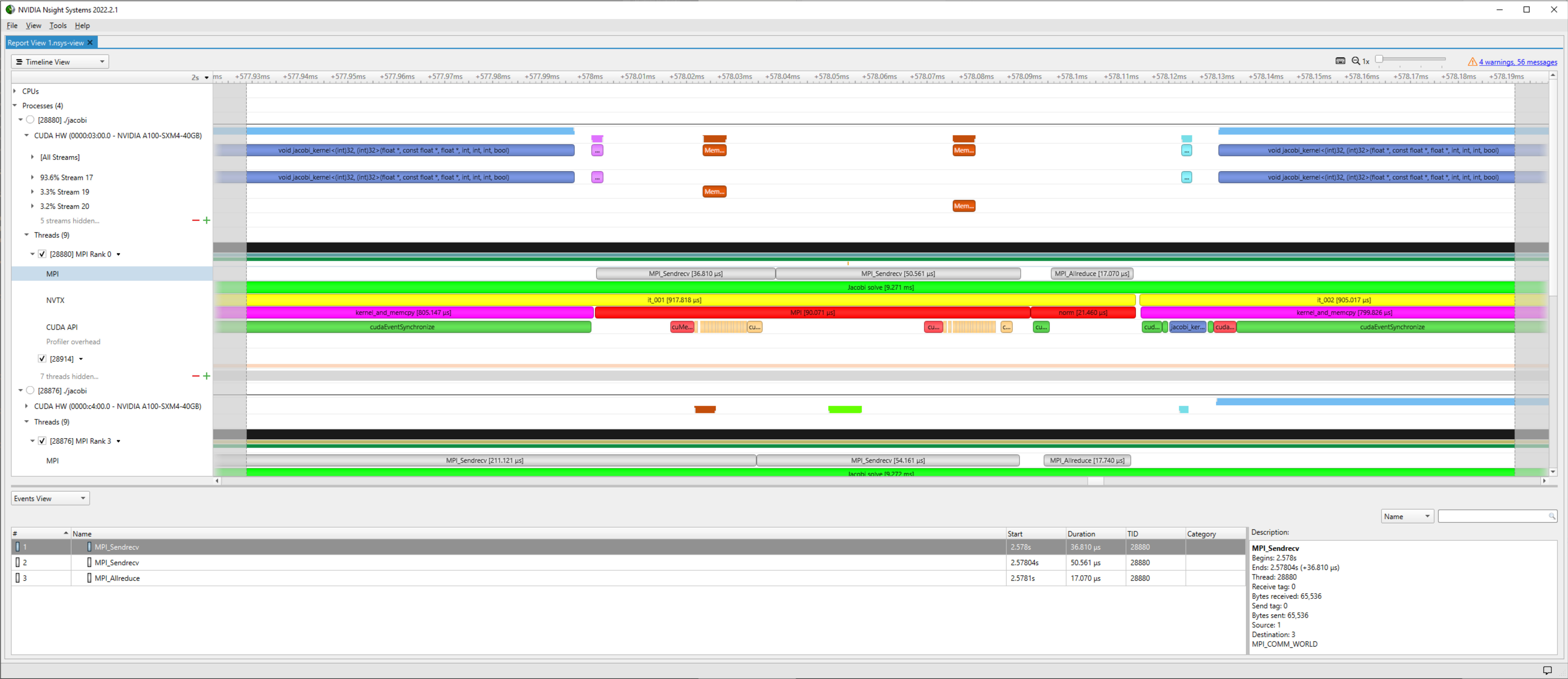


# Optimization Techniques for Multi-GPU Applications

Jiri Kraus, NVIDIA, Principal Devtech Compute

# MULTI GPU JACOBI NSIGHT SYSTEMS TIMELINE

MPI 8 NVIDIA A100 40GB on JUWELS Booster



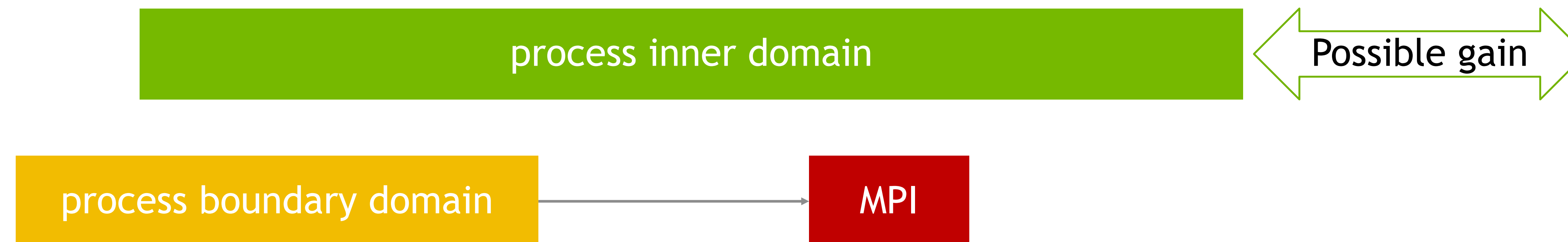


# COMMUNICATION + COMPUTATION OVERLAP

No Overlap

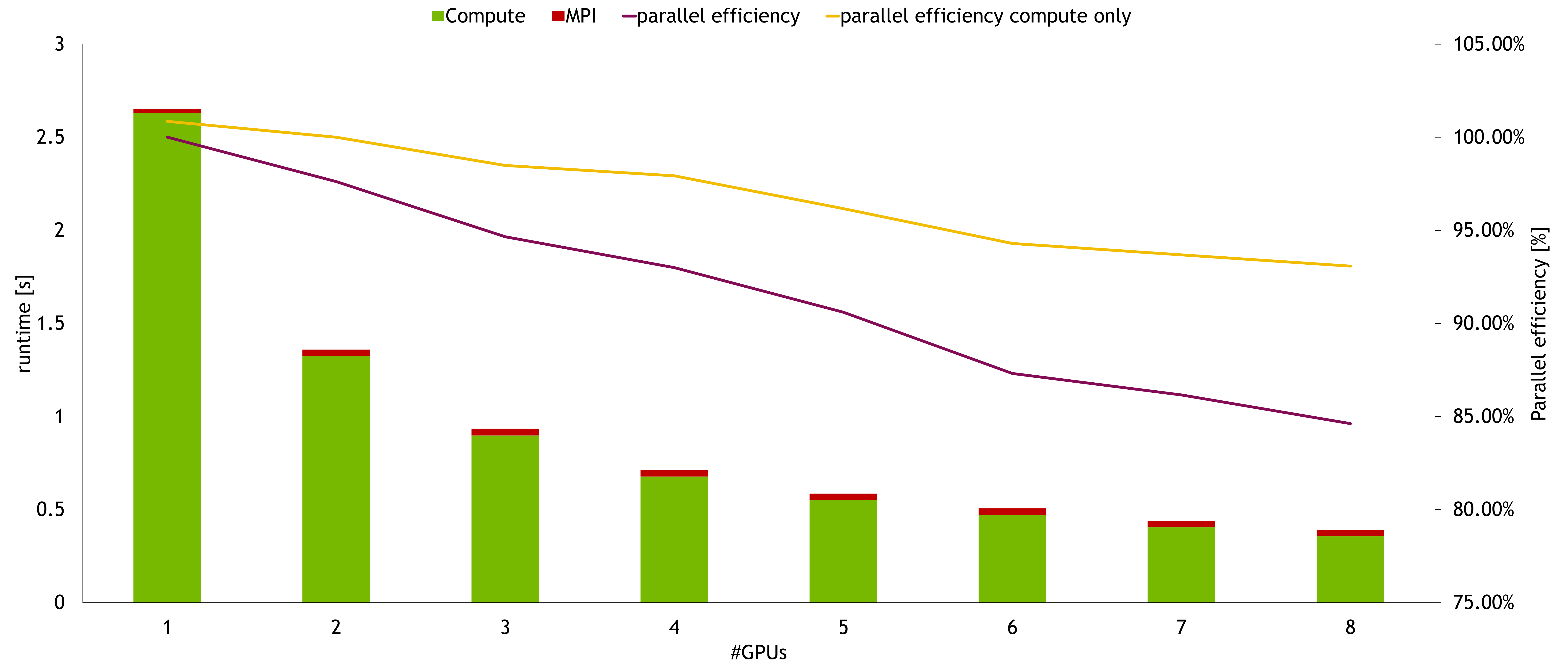


Overlap



# COMMUNICATION + COMPUTATION OVERLAP

ParaStationMPI 5.4.10-1 - JUWELS Booster - NVIDIA A100 40 GB - Jacobi on 17408x17408



Source: <https://github.com/NVIDIA/multi-gpu-programming-models>  
JUWELS Booster: <https://apps.fz-juelich.de/jsc/hps/juwels/booster-overview.html>

# MPI COMMUNICATION + COMPUTATION OVERLAP

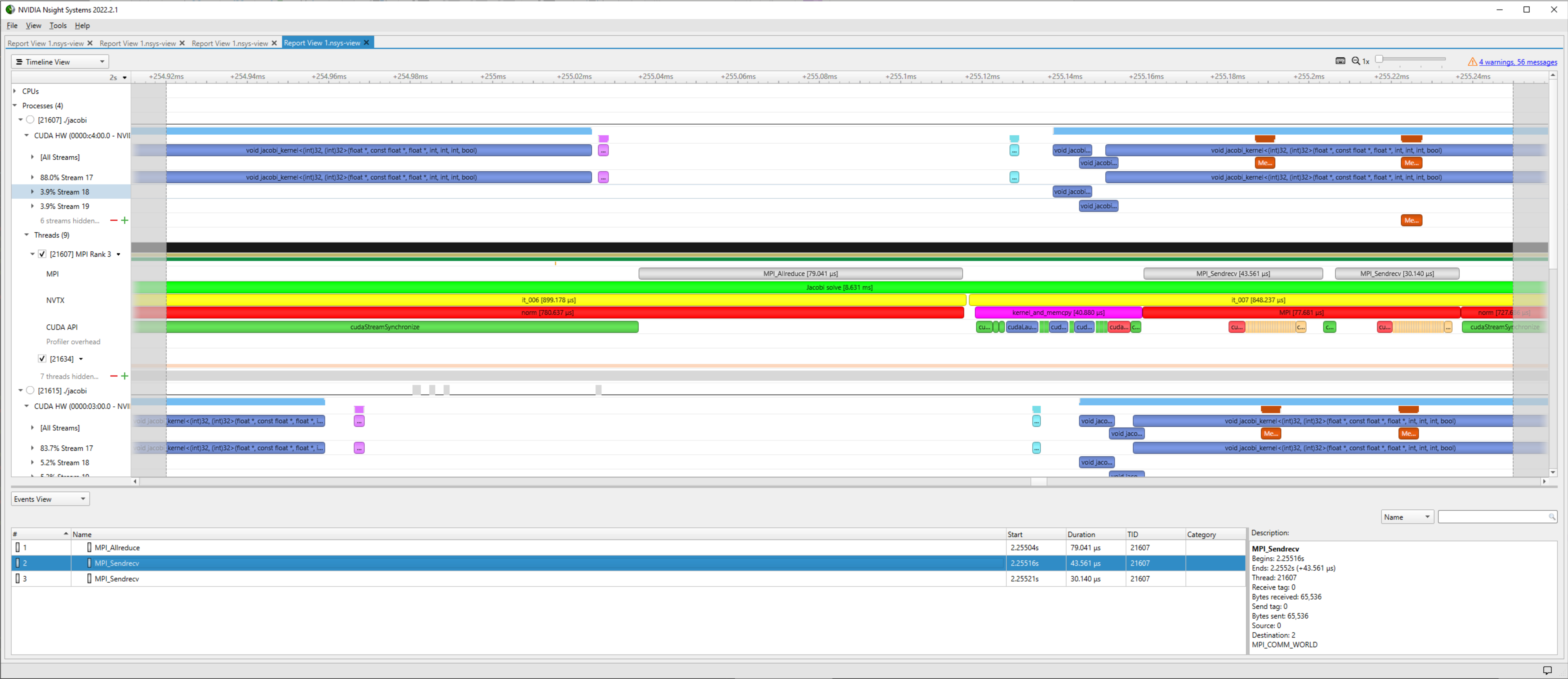
```
launch_jacobi_kernel(a_new, a, l2_norm_d, iy_start, (iy_start + 1), nx, push_top_stream);
launch_jacobi_kernel(a_new, a, l2_norm_d, (iy_end - 1), iy_end, nx, push_bottom_stream);
launch_jacobi_kernel(a_new, a, l2_norm_d, (iy_start + 1), (iy_end - 1), nx, compute_stream);

const int top = rank > 0 ? rank - 1 : (size - 1);
const int bottom = (rank + 1) % size;

CUDA_RT_CALL(cudaStreamSynchronize(push_top_stream));
MPI_CALL(MPI_Sendrecv(a_new + iy_start * nx, nx, MPI_REAL_TYPE, top, 0,
                     a_new + (iy_end * nx), nx, MPI_REAL_TYPE, bottom, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE));
CUDA_RT_CALL(cudaStreamSynchronize(push_bottom_stream));
MPI_CALL(MPI_Sendrecv(a_new + (iy_end - 1) * nx, nx, MPI_REAL_TYPE, bottom, 0,
                     a_new, nx, MPI_REAL_TYPE, top, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE));
```

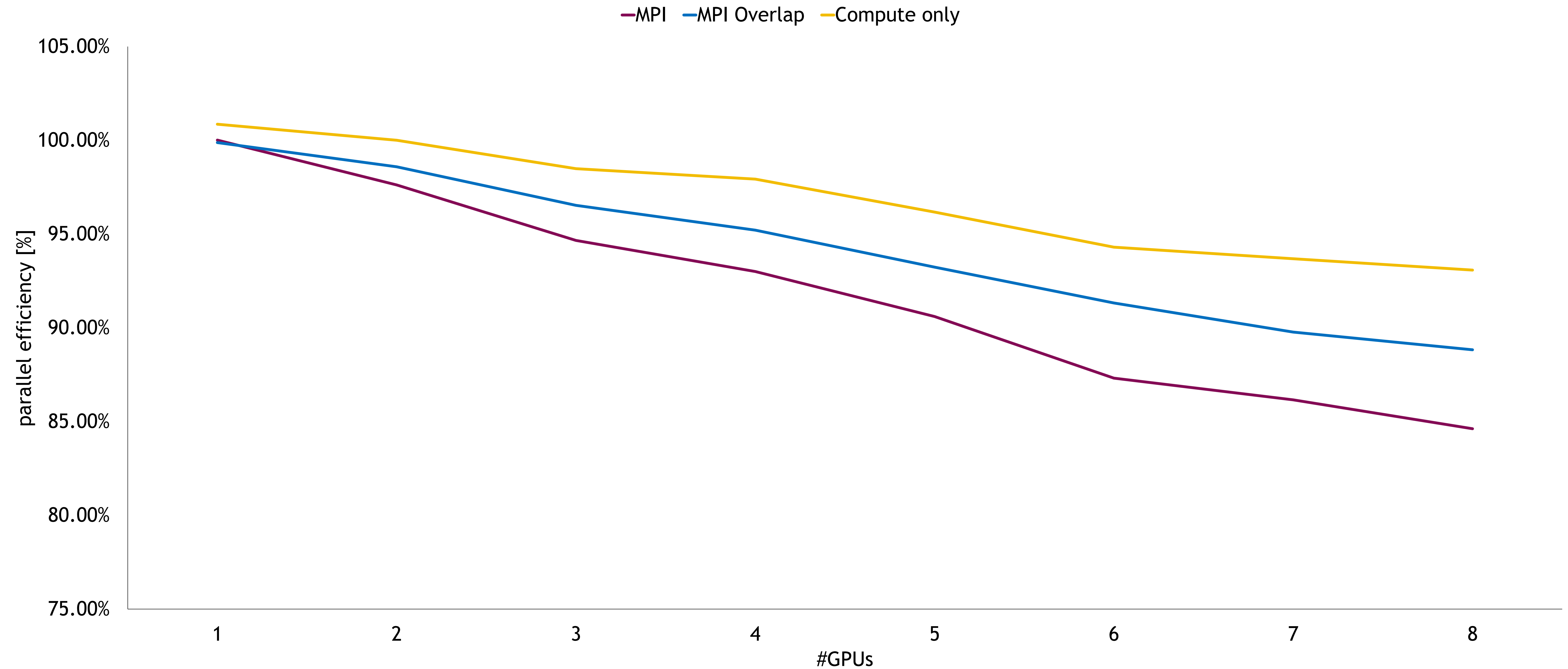
# MULTI GPU JACOBI NSIGHT SYSTEMS TIMELINE

## MPI Overlap 8 NVIDIA A100 40GB on JUWELS Booster



# COMMUNICATION + COMPUTATION OVERLAP

ParaStationMPI 5.4.10-1 - JUWELS Booster - NVIDIA A100 40 GB - Jacobi on 17408x17408




Source: <https://github.com/NVIDIA/multi-gpu-programming-models>  
JUWELS Booster: <https://apps.fz-juelich.de/jsc/hps/juwels/booster-overview.html>



# MPI COMMUNICATION + COMPUTATION OVERLAP

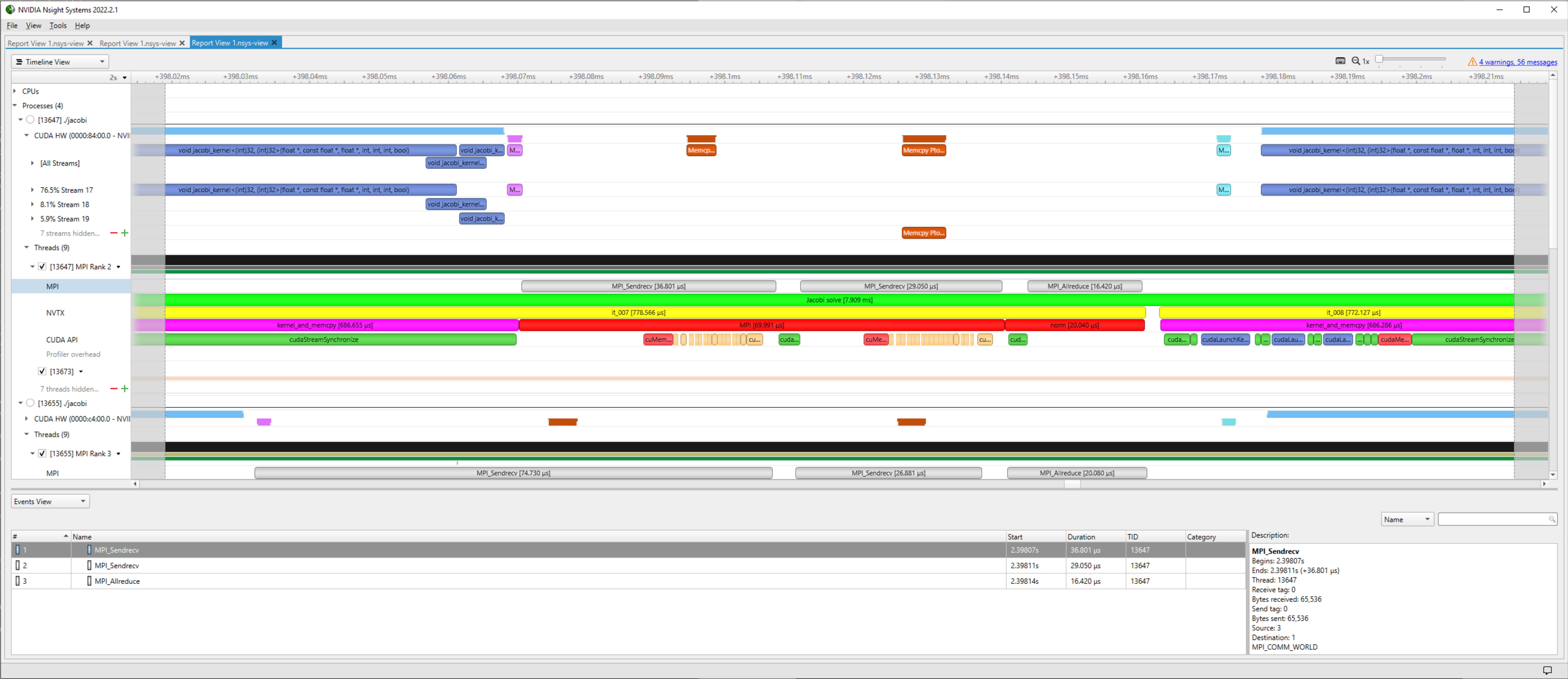
```
launch_jacobi_kernel(a_new, a, l2_norm_d, (iy_start + 1), (iy_end - 1), nx, compute_stream);  
launch_jacobi_kernel(a_new, a, l2_norm_d, iy_start, (iy_start + 1), nx, push_top_stream);  
launch_jacobi_kernel(a_new, a, l2_norm_d, (iy_end - 1), iy_end, nx, push_bottom_stream);  
launch_jacobi_kernel(a_new, a, l2_norm_d, (iy_start + 1), (iy_end - 1), nx, compute_stream);  
  
const int top = rank > 0 ? rank - 1 : (size - 1);  
const int bottom = (rank + 1) % size;  
  
CUDA_RT_CALL(cudaStreamSynchronize(push_top_stream));  
MPI_CALL(MPI_Sendrecv(a_new + iy_start * nx, nx, MPI_REAL_TYPE, top, 0,  
    a_new + (iy_end * nx), nx, MPI_REAL_TYPE, bottom, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE));  
CUDA_RT_CALL(cudaStreamSynchronize(push_bottom_stream));  
MPI_CALL(MPI_Sendrecv(a_new + (iy_end - 1) * nx, nx, MPI_REAL_TYPE, bottom, 0,  
    a_new, nx, MPI_REAL_TYPE, top, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE));
```





# MULTI GPU JACOBI NSIGHT SYSTEMS TIMELINE

## MPI Overlap 8 NVIDIA A100 40GB on JUWELS Booster

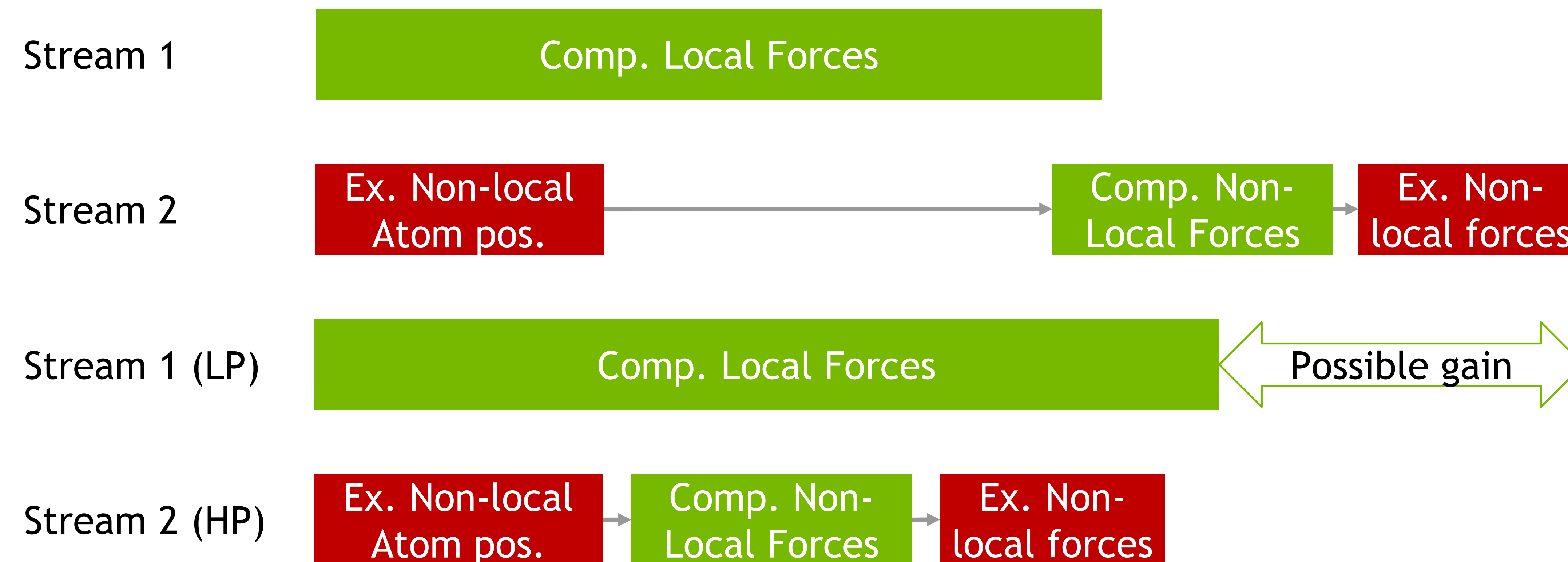


# HIGH PRIORITY STREAMS

Improve scalability with high priority streams (available on CC 3.5+)

```
cudaStreamCreateWithPriority ( cudaStream_t* pStream, unsigned int flags, int priority )
```

Use-case: MD-Simulations



# MPI COMMUNICATION + COMPUTATION OVERLAP

with high priority streams

```
int leastPriority = 0;
int greatestPriority = leastPriority;
CUDA_RT_CALL(cudaDeviceGetStreamPriorityRange(&leastPriority, &greatestPriority));

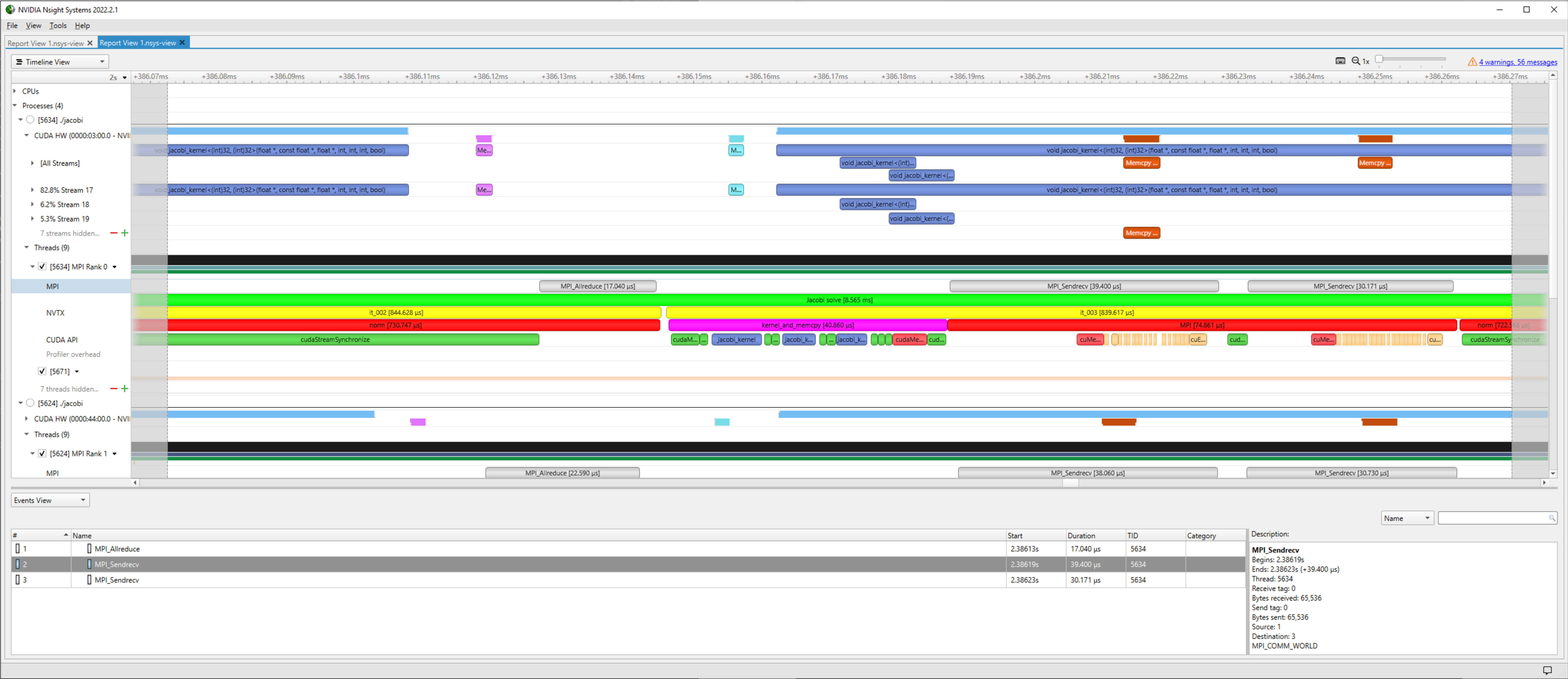
cudaStream_t compute_stream;
cudaStream_t push_top_stream;
cudaStream_t push_bottom_stream;

CUDA_RT_CALL(cudaStreamCreateWithPriority(&compute_stream, cudaStreamDefault, leastPriority));
CUDA_RT_CALL(cudaStreamCreateWithPriority(&push_top_stream, cudaStreamDefault, greatestPriority));
CUDA_RT_CALL(cudaStreamCreateWithPriority(&push_bottom_stream, cudaStreamDefault, greatestPriority));
```



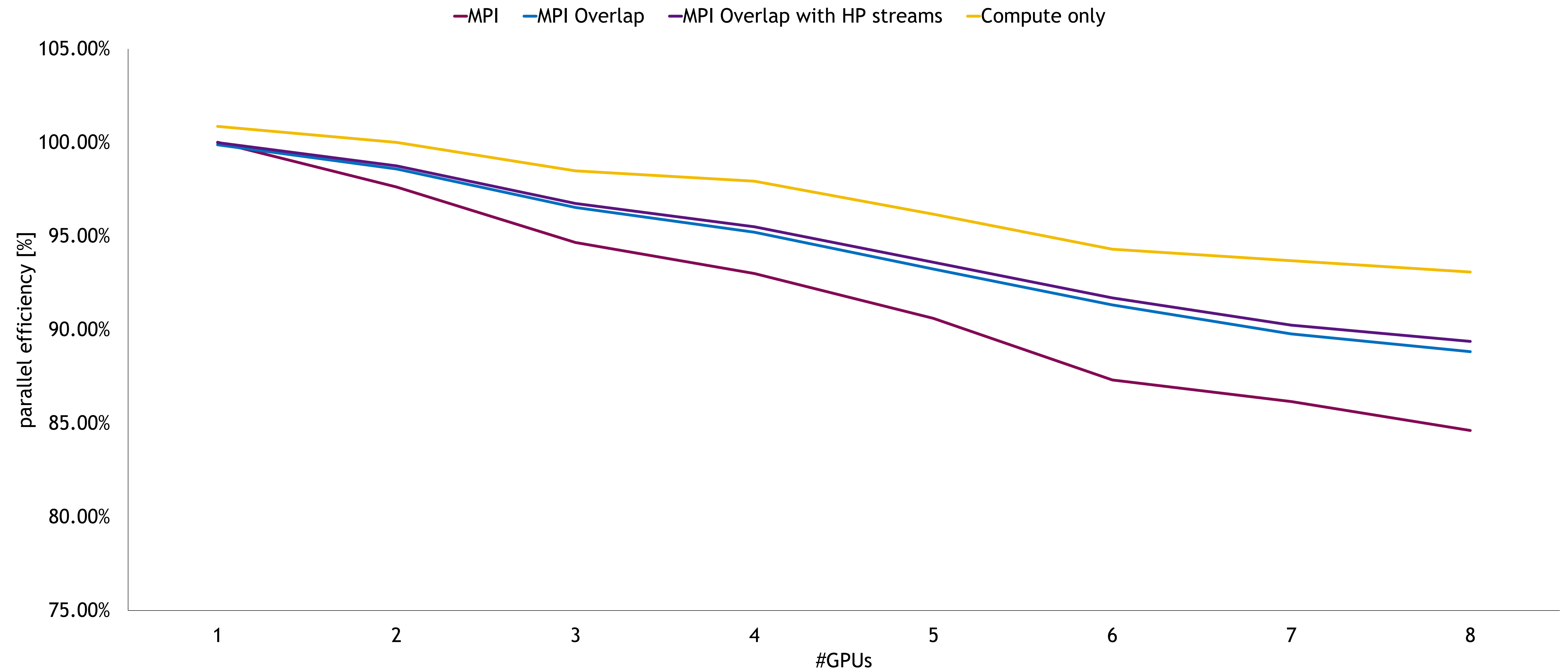
# MULTI GPU JACOBI NSIGHT SYSTEMS TIMELINE

## MPI Overlap 8 NVIDIA A100 40GB on JUWELS Booster



# COMMUNICATION + COMPUTATION OVERLAP

ParaStationMPI 5.4.10-1 - JUWELS Booster - NVIDIA A100 40 GB - Jacobi on 17408x17408



# CUDA-AWARE MPI

Example:

MPI Rank 0 MPI\_Send from GPU Buffer

MPI Rank 1 MPI\_Recv to GPU Buffer

Show how CUDA+MPI works in principle

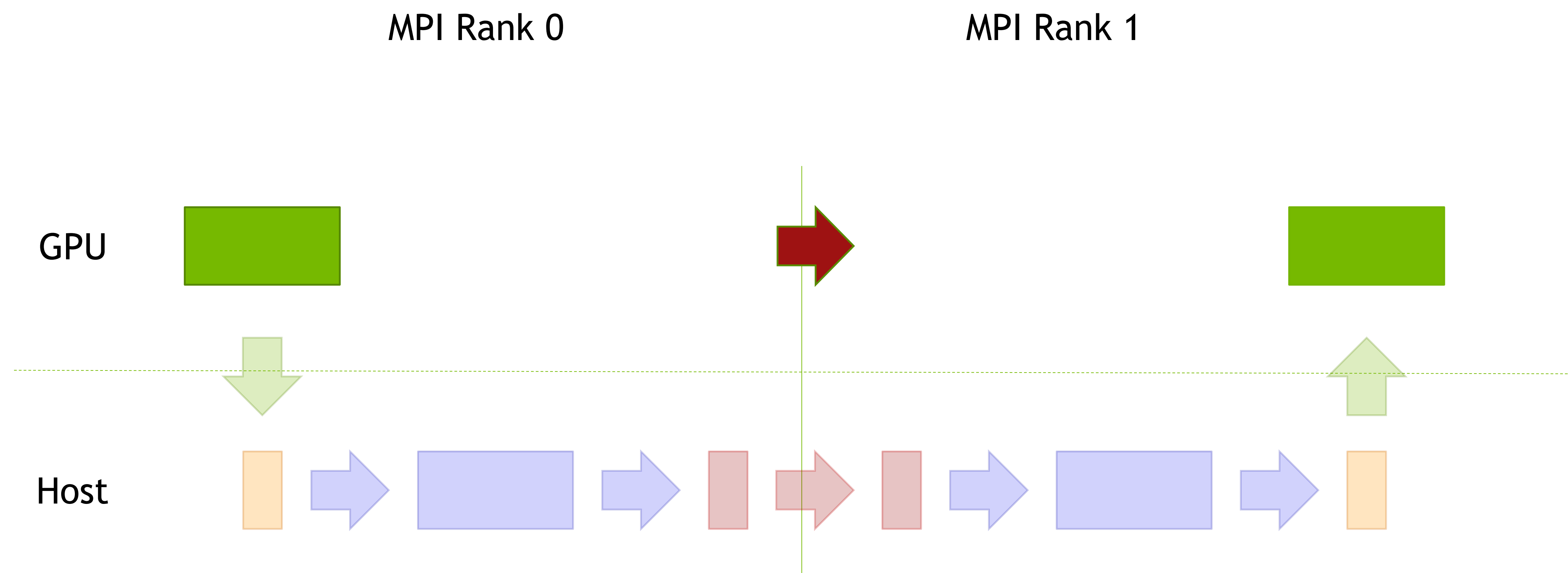
Depending on the MPI implementation, message size, system setup, ... situation might be different

Two GPUs in two nodes



# GPU TO REMOTE GPU

CUDA-aware MPI with support for GPUDirect RDMA



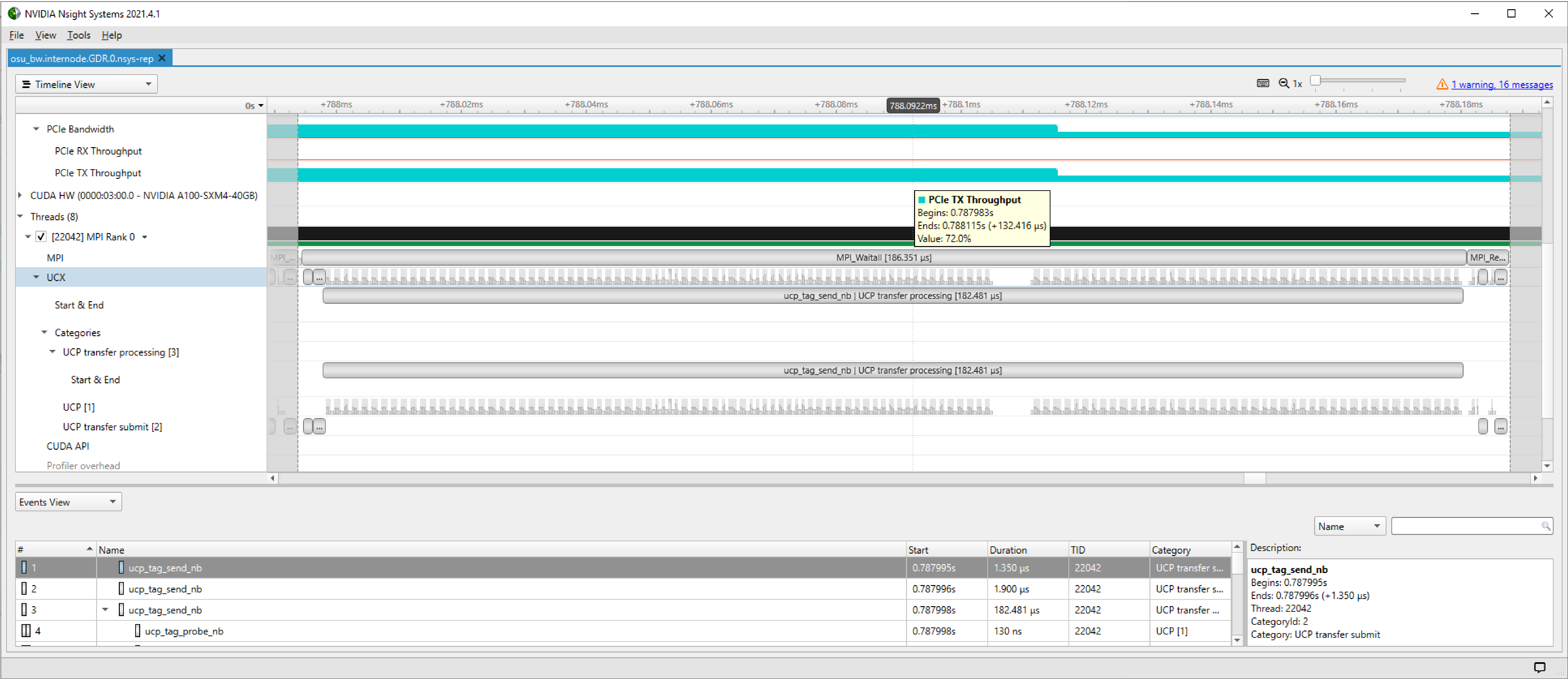
```
MPI_Send(s_buf_d, size, MPI_BYTE, 1, tag, MPI_COMM_WORLD);
```

```
MPI_Recv(r_buf_d, size, MPI_BYTE, 0, tag, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

# OSU\_BW NSIGHT SYSTEMS TIMELINE

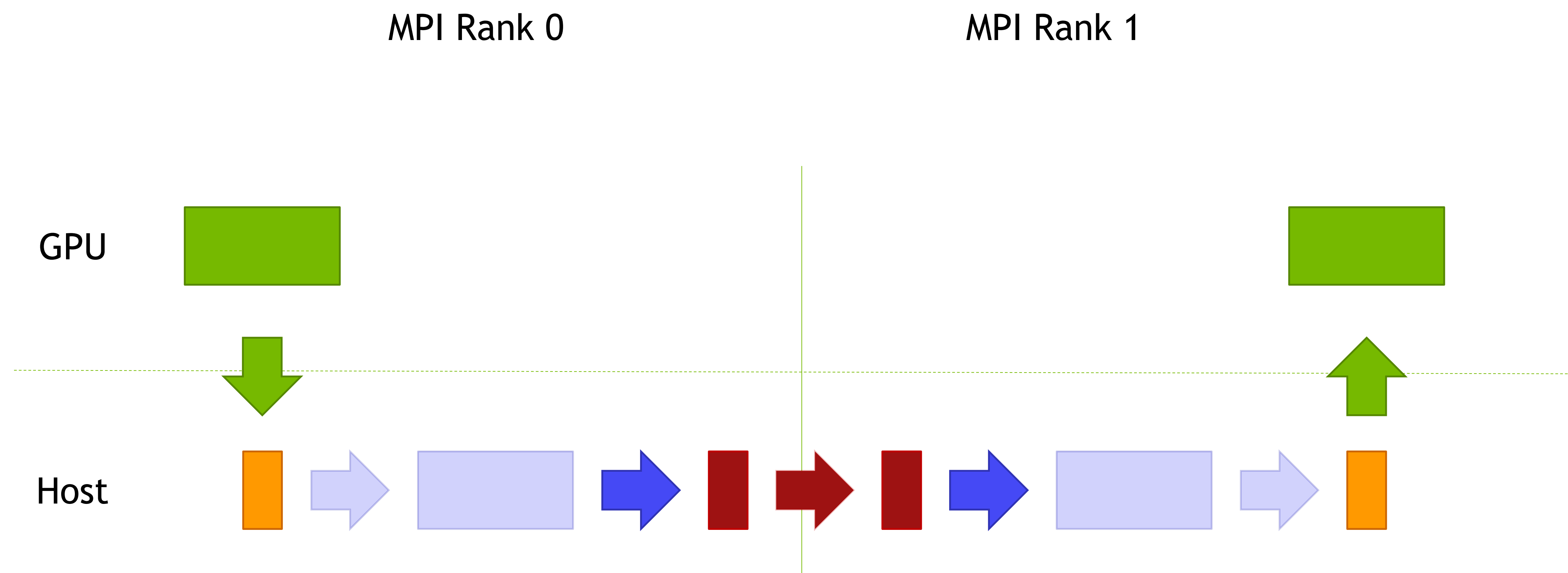
Internode with GPUDirect RDMA on JUWELS Booster

```
nsys profile --gpu-metrics-device=0 --trace=mpi,ucx,cuda -o osu_bw.internode.GDR.%q{SLURM_PROCID}
```



# GPU TO REMOTE GPU

CUDA-aware MPI without support for GPUDirect



```
MPI_Send(s_buf_d, size, MPI_BYTE, 1, tag, MPI_COMM_WORLD);
```

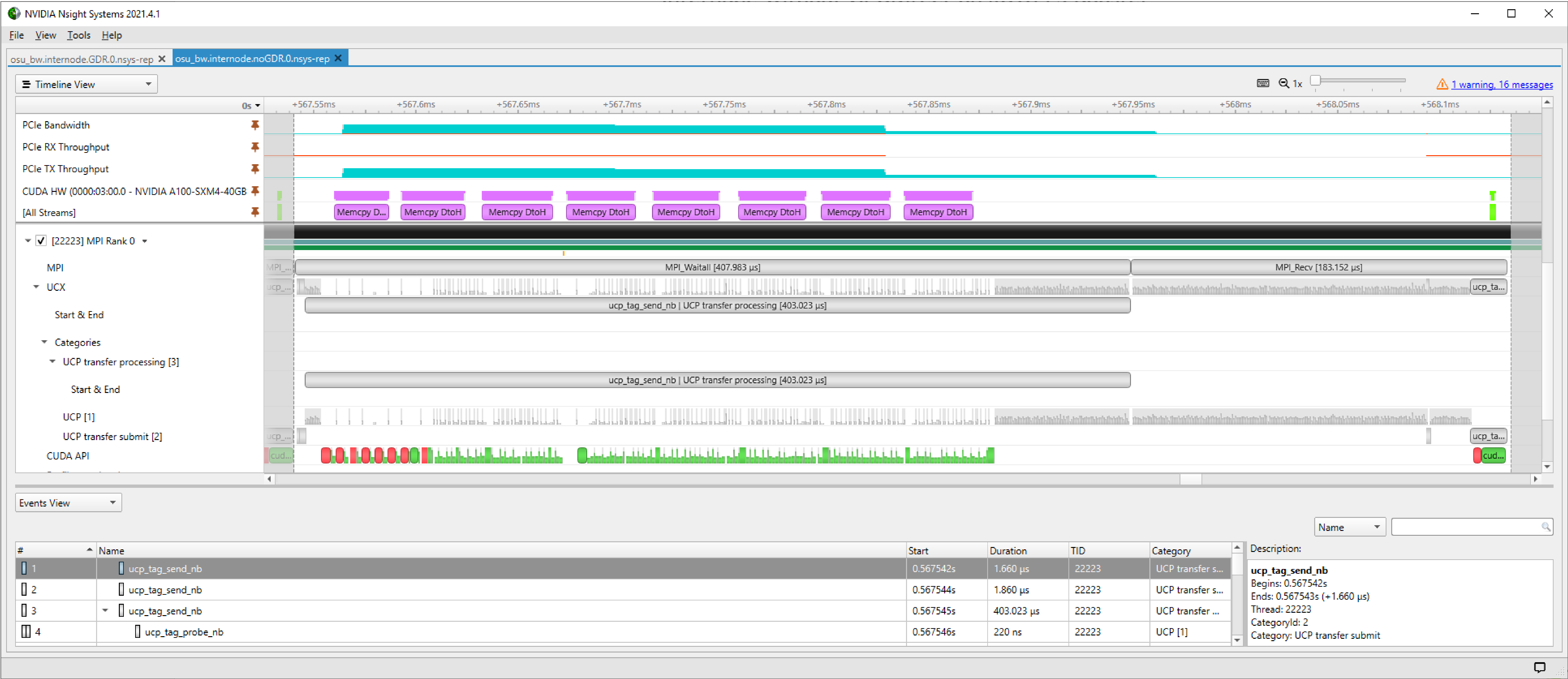
```
MPI_Recv(r_buf_d, size, MPI_BYTE, 0, tag, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```



# OSU\_BW NSIGHT SYSTEMS TIMELINE

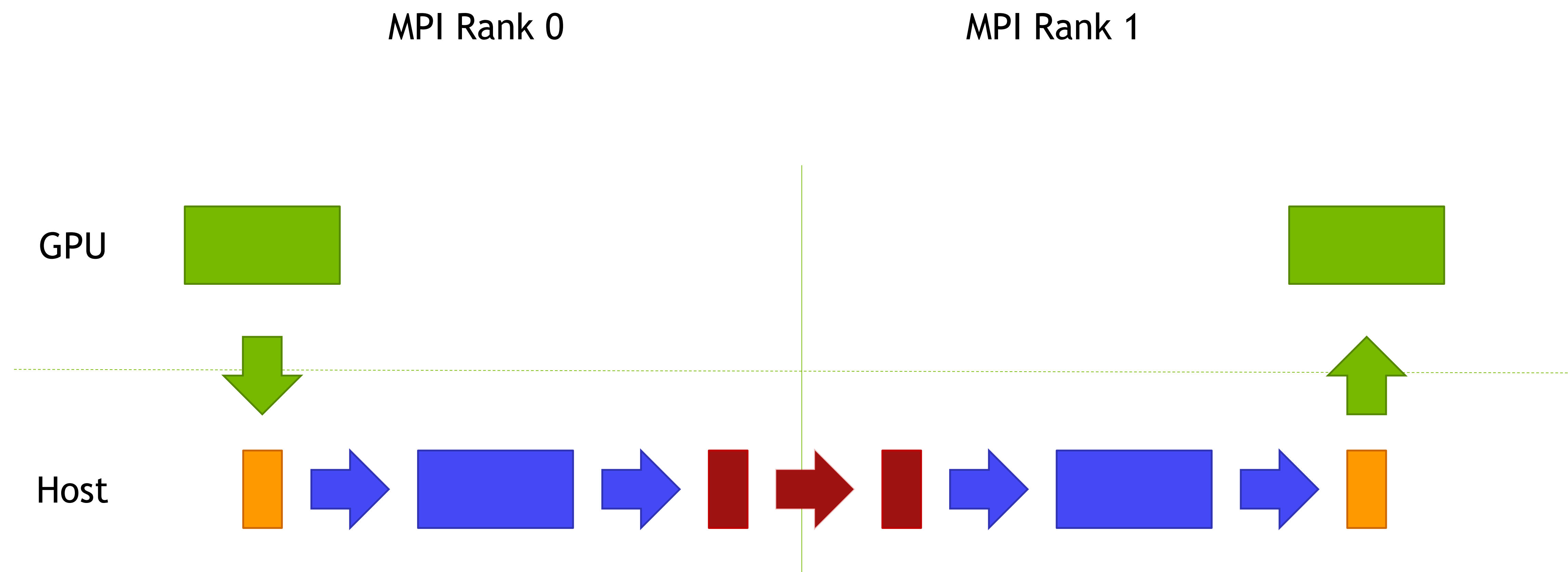
Internode without GPUDirect on JUWELS Booster

```
nsys profile --gpu-metrics-device=0 --trace=mpi,ucx,cuda -o osu_bw.internode.noGDR.%q{SLURM_PROCID}
```



# GPU TO REMOTE GPU

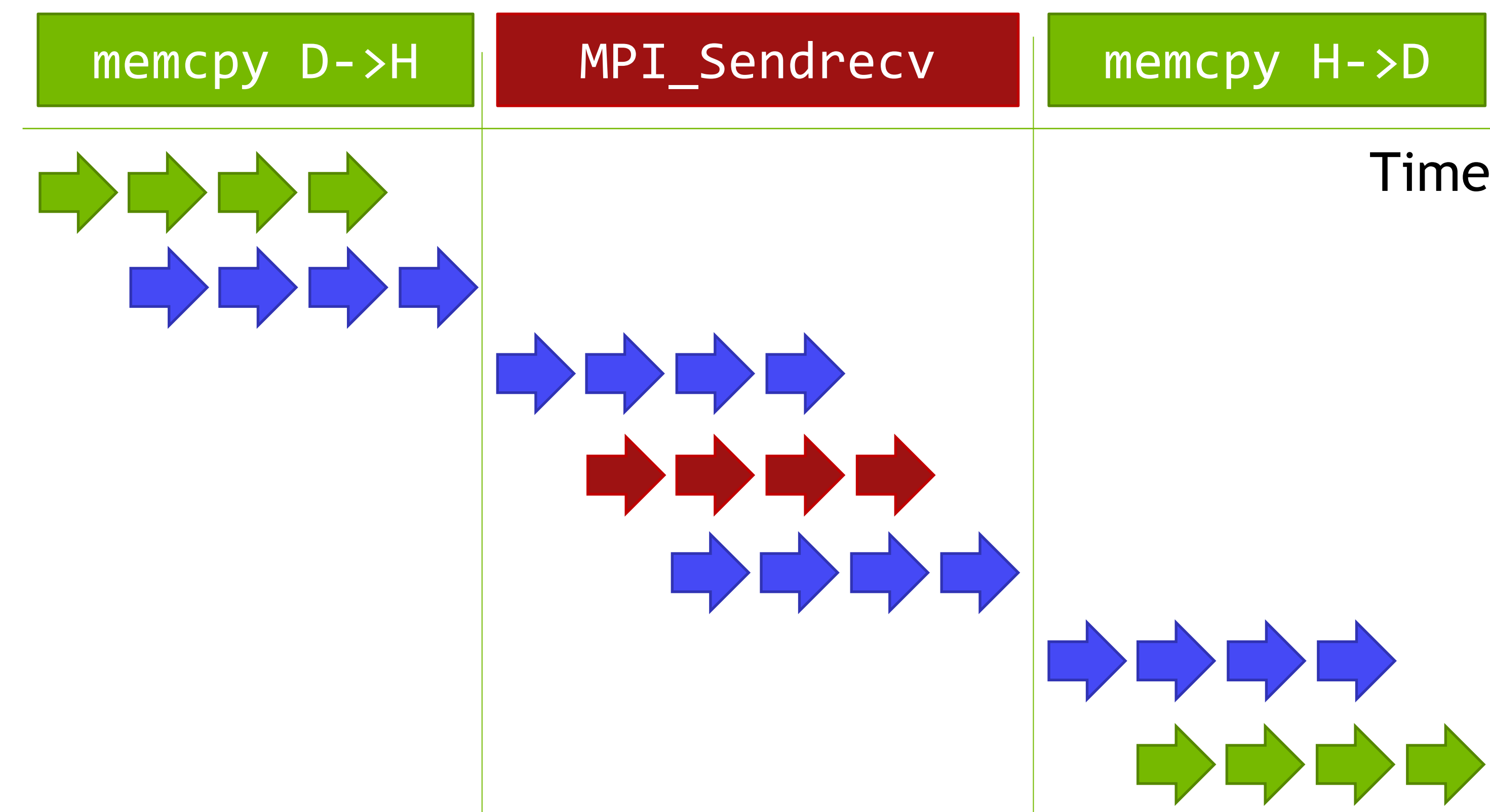
MPI without CUDA support



```
cudaMemcpy(s_buf_h, s_buf_d, size, cudaMemcpyDeviceToHost);  
MPI_Send(s_buf_d, size, MPI_BYTE, 1, tag, MPI_COMM_WORLD);  
  
MPI_Recv(r_buf_d, size, MPI_BYTE, 0, tag, MPI_COMM_WORLD, MPI_STATUS_IGNORE);  
cudaMemcpy(r_buf_d, r_buf_h, size, cudaMemcpyHostToDevice);
```

# GPU TO REMOTE GPU

MPI without CUDA support

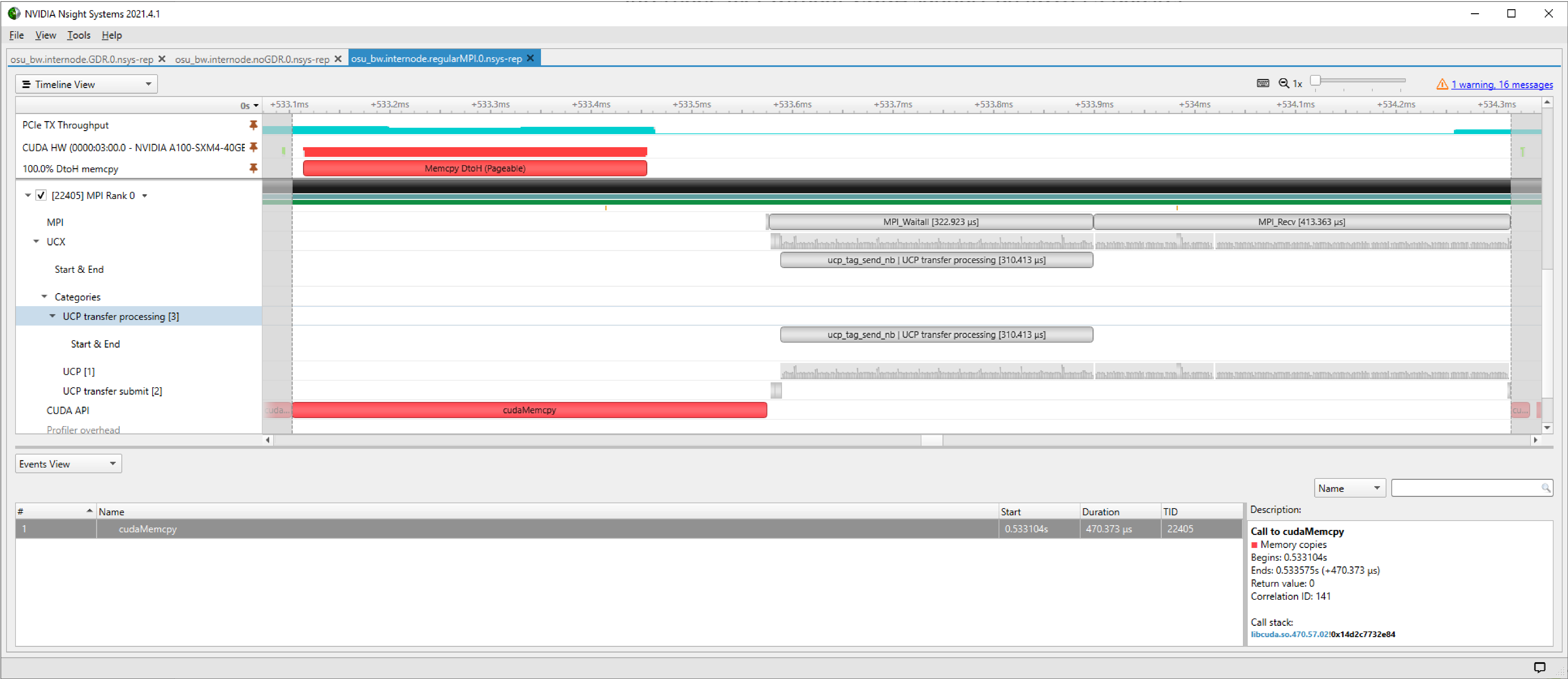




# OSU\_BW NSIGHT SYSTEMS TIMELINE

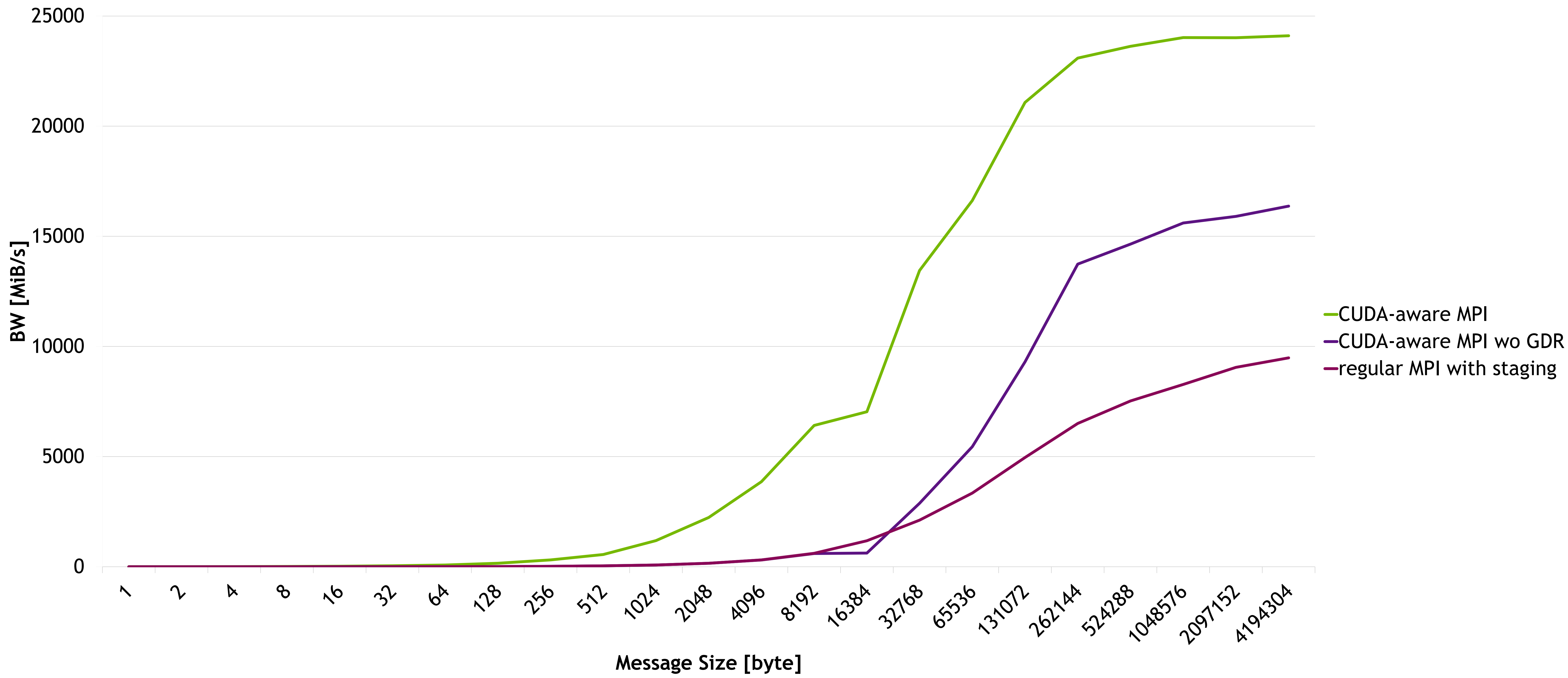
Internode MPI without CUDA support on JUWELS Booster

```
nsys profile --gpu-metrics-device=0 --trace=mpi,ucx,cuda -o osu_bw.internode.noCUDAMPI.%q{SLURM_PROCID}
```



# PERFORMANCE RESULTS GPUDIRECT RDMA

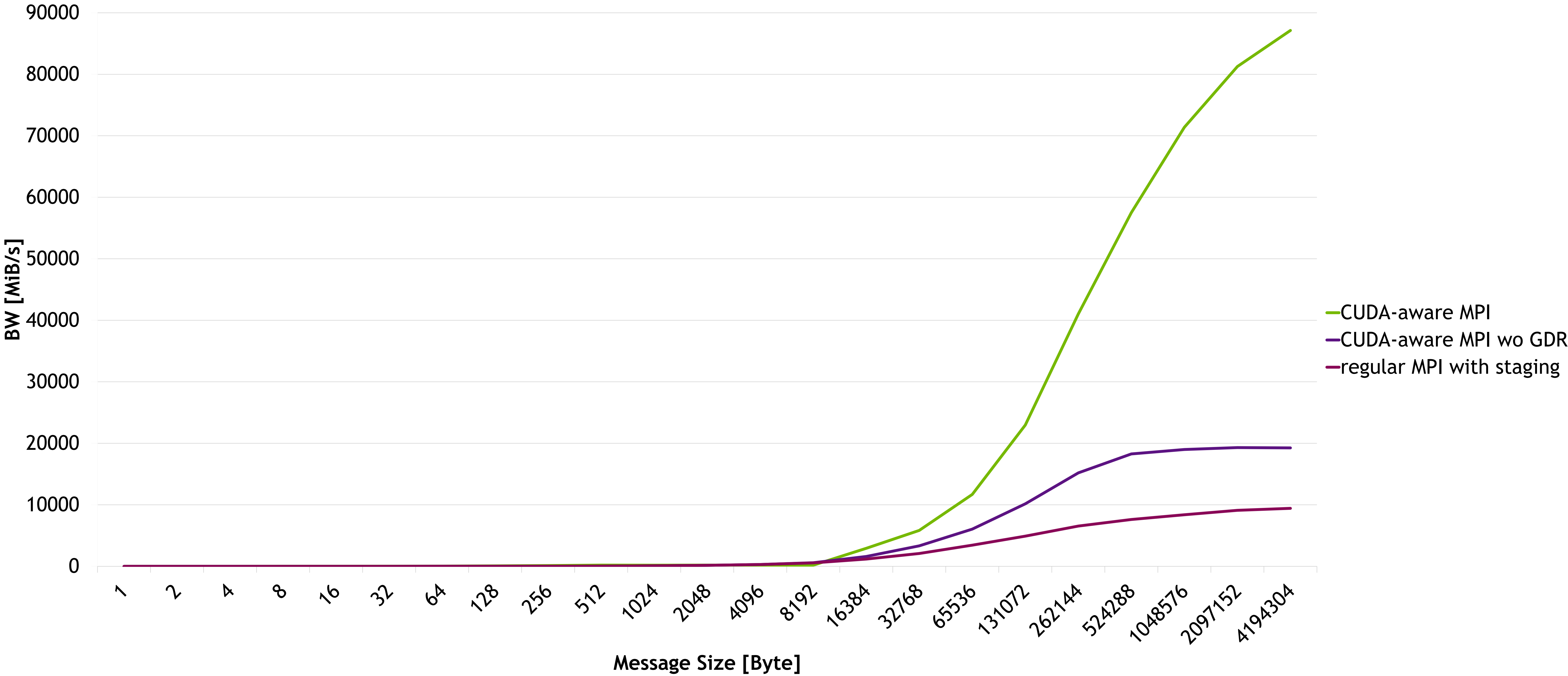
OpenMPI 4.1.0RC1 + UCX 1.9.0 on JUWELS Booster



Latency (1 byte) 4.27 us 24.56 us 25.64 us

# PERFORMANCE RESULTS GPUDIRECT P2P

OpenMPI 4.1.0RC1 + UCX 1.9.0 on JUWELS Booster



Latency (1 byte) 2.45 us 22.01 us 23.50 us



# UCX TIPS AND TRICKS

Check setting and knobs with ucx\_info

```
$ ucx_info -caf | grep -B9 UCX_RNDV_SCHEME
```

```
#
```

```
# Communication scheme in RNDV protocol.
```

```
# get_zcopy - use get_zcopy scheme in RNDV protocol.
```

```
# put_zcopy - use put_zcopy scheme in RNDV protocol.
```

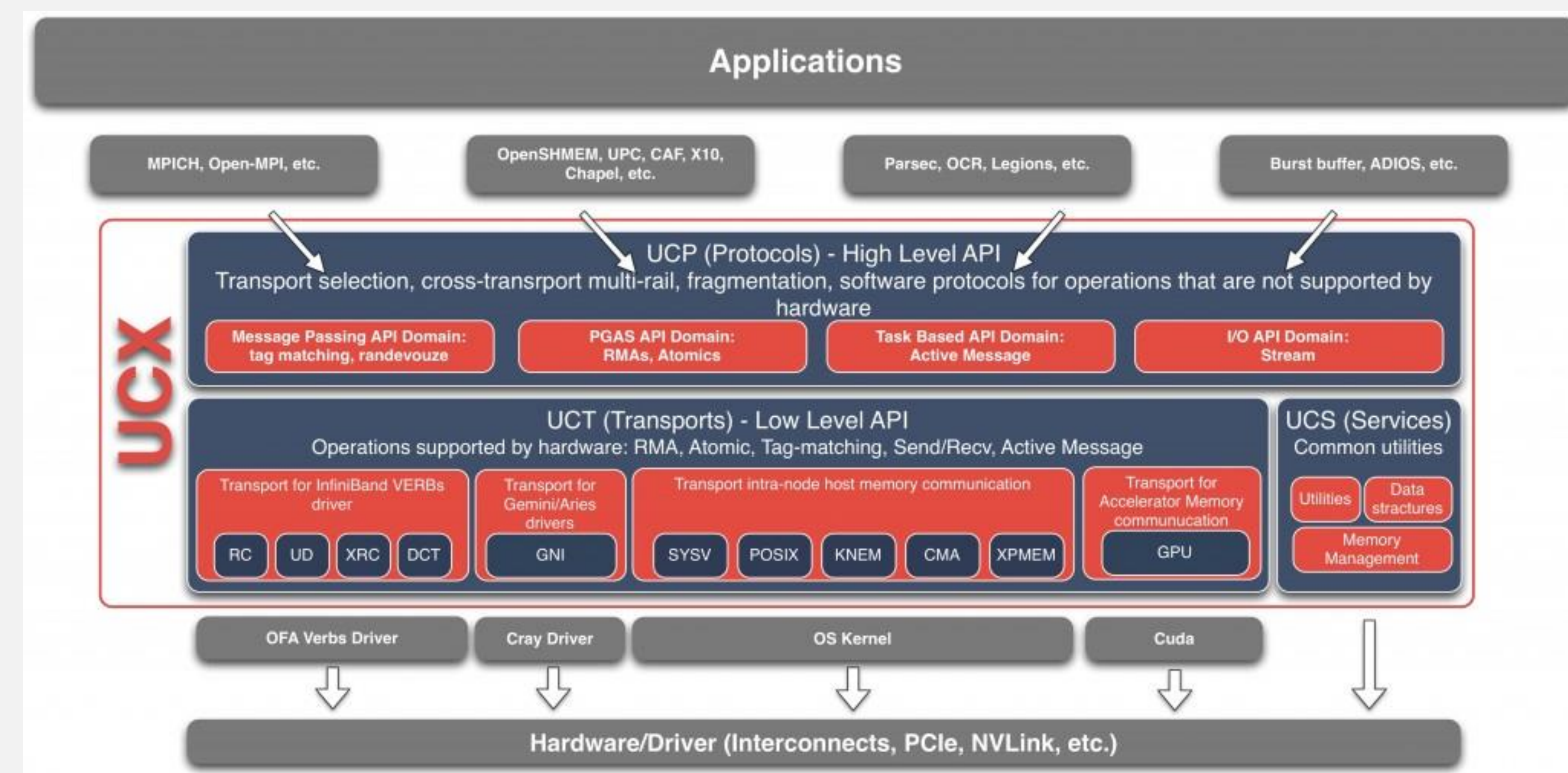
```
# auto - runtime automatically chooses optimal  
scheme to use.
```

```
#
```

```
# syntax: [get_zcopy|put_zcopy|auto]
```

```
#
```

```
UCX_RNDV_SCHEME=auto
```



© 2022 Unified Communication X: <https://openucx.org/license/>



# UCX TIPS AND TRICKS

Enable logging to see what is going on

UCX\_LOG\_LEVEL=data UCX\_LOG\_FILE=log-%h-%p helpful to check for used protocols and selected HCAs:

```
[1605706306.970537] [jwb1238:7263 :0]      ucp_worker.c:1627 UCX  INFO  ep_cfg[0]: tag(cuda_copy/cuda);  
rma(gdr_copy/cuda);
```

```
[1605706306.972721] [jwb1238:7263 :0]      ucp_worker.c:1627 UCX  INFO  ep_cfg[1]: tag(self/memory  
rc_mlx5/mlx5_1:1 cma/memory cuda_copy/cuda);
```

```
[1605706306.997849] [jwb1238:7263 :1]      ucp_worker.c:1627 UCX  INFO  ep_cfg[2]: tag(rc_mlx5/mlx5_1:1);
```

# UCX TIPS AND TRICKS

<https://github.com/openucx/ucx/wiki/UCX-environment-parameters>

**UCX\_NET\_DEVICES:** To select HCA for optimal GPU-HCA affinity, should not be necessary with UCX 1.9 or newer

**UCX\_TLS:** Select transports to use, default: all

cuda is an alias for: cuda\_copy, cuda\_ipc, gdr\_copy

To run without any GPUDirect flavor set UCX\_TLS to only include cuda\_copy, e.g. UCX\_TLS=rc,sm,cuda\_copy and UCX\_IB\_GPU\_DIRECT\_RDMA=no (rc transport uses GPUDirect RDMA otherwise).

Parastation MPI also has PSP\_CUDA\_ENFORCE\_STAGING=1.

**UCX\_MEMTYPE\_CACHE:** Set to n to disable mem type cache. Sometimes necessary if the CUDA runtime is linked statically!

# SUMMARY

- Asynchronously computing on the GPU while MPI communication allows to hide MPI communication times
- Using high priority streams some CUDA API overheads can be also hidden
- GPUDirect can provide significant performance improvements both inter and intra node
- Knowing UCX performance tuning knobs is important when working with CUDA-aware MPI implementations like OpenMPI and Parastation MPI built on UCX.