



# SUMMARY AND ADVANCED TOPICS ISC23 TUTORIAL SESSION 11

21 May 2023 | Andreas Herten | Jülich Supercomputing Centre, Forschungszentrum Jülich

# Overview

## Summary

1L: JUWELS Booster

2L: MPI-Distributed GPU Computing

4L: Performance/Debugging Tools

5L: Optimization Techniques

7L: NCCL, NVSHMEM

9L: Device-Initiated NVSHMEM

*More: Other Languages/Models*

OpenACC, OpenMP; Kokkos

Python

*More: In-Network Computing*

Concept

Libraries

Other Vendors

Summary, Conclusion

# Summary

## *1L: JUWELS Booster*

# JUWELS Booster Overview

## Node Configuration

Arch Atos Bull Sequana XH2000

CPU 2 × AMD EPYC 7402:

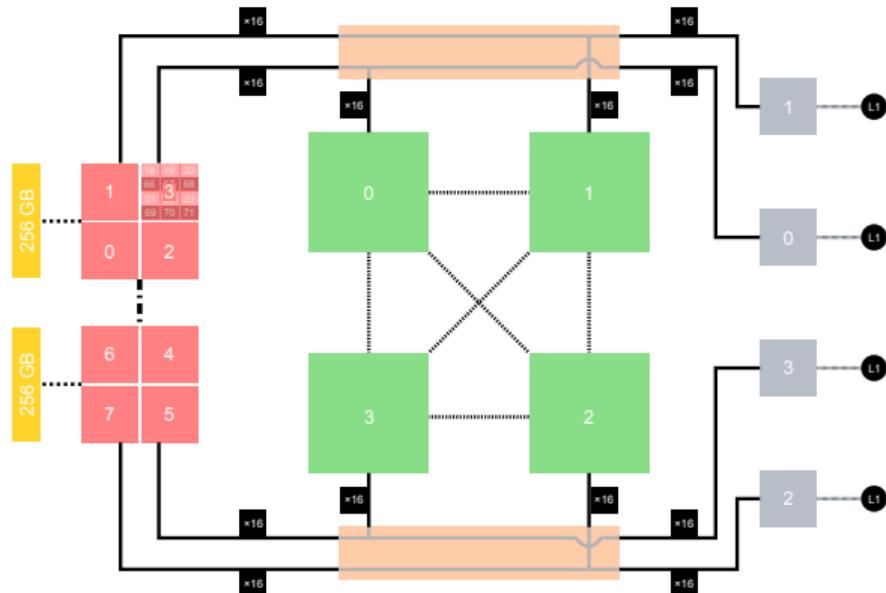
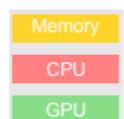
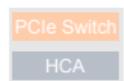
2Socket × 24Core × 2SMT,  
2 × 256 GB DDR4-3200 RAM;  
NPS-4

GPU 4 × NVIDIA A100 40 GB, NVLink3

HCA 4 × Mellanox HDR200  
(200 Gbit/s) InfiniBand  
ConnectX 6

etc 2 × PCIe Gen 4 switch

→ Many affinities



# JUWELS Booster Overview

## Node Configuration

Arch Atos Bull Sequana XH2000

CPU 2 × AMD EPYC 7402:

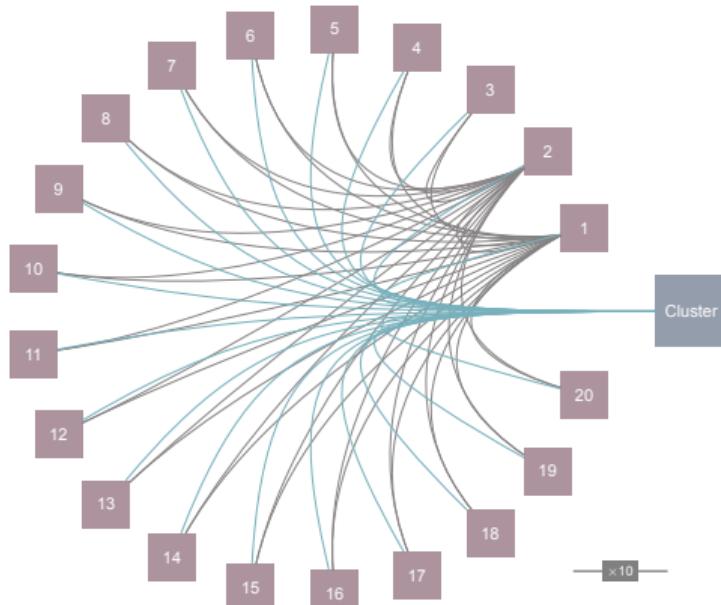
2<sub>Socket</sub> × 24<sub>Core</sub> × 2<sub>SMT</sub>,  
2 × 256 GB DDR4-3200 RAM;  
NPS-4

GPU 4 × NVIDIA A100 40 GB, NVLink3

HCA 4 × Mellanox HDR200  
(200 Gbit/s) InfiniBand  
ConnectX 6

etc 2 × PCIe Gen 4 switch

→ Many affinities



# Summary

2L: MPI-Distributed GPU Computing



## CUDA-aware MPI

CUDA-aware MPI allows you to use Pointers to GPU-Memory as source and destination

```
//MPI rank 0  
MPI_Send(s_buf_d, n, MPI_BYTE, size-1, tag, MPI_COMM_WORLD);  
  
//MPI size-1  
MPI_Recv(r_buf_d, n, MPI_BYTE, 0, tag, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

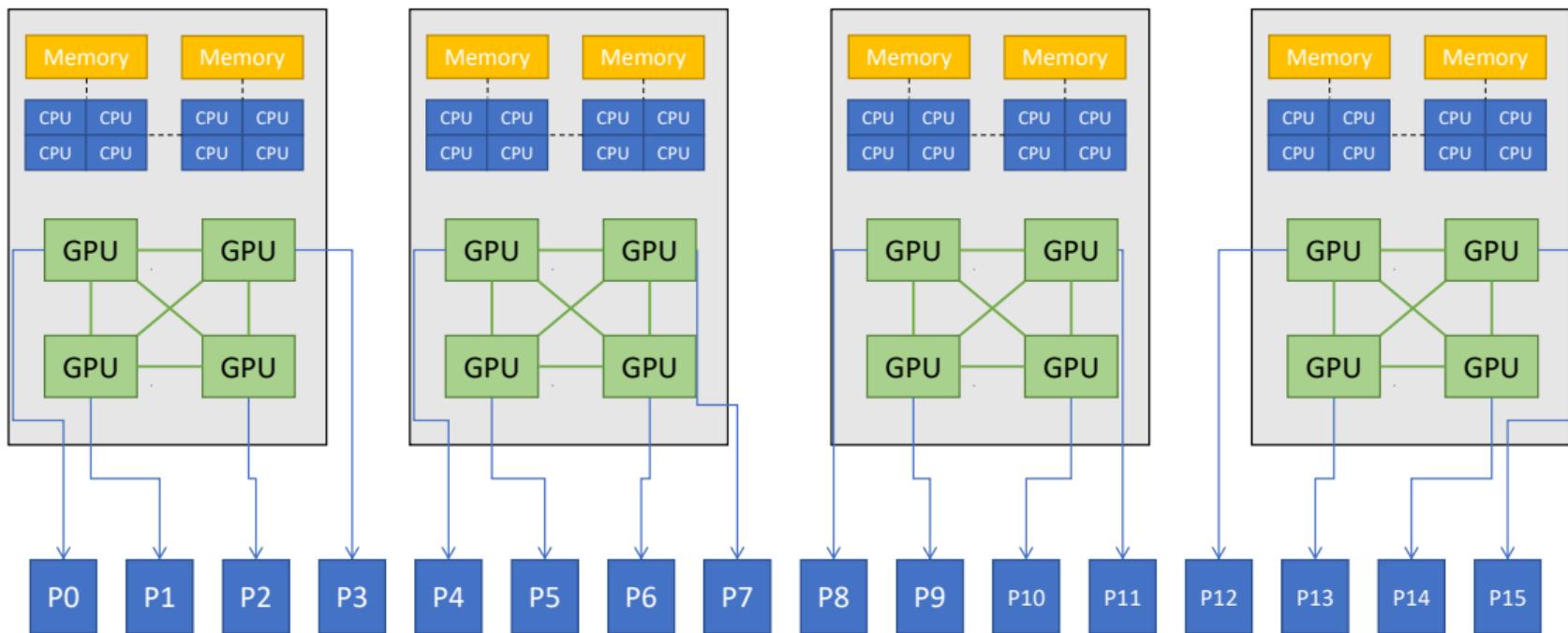


Pointer to  
GPU  
memory!



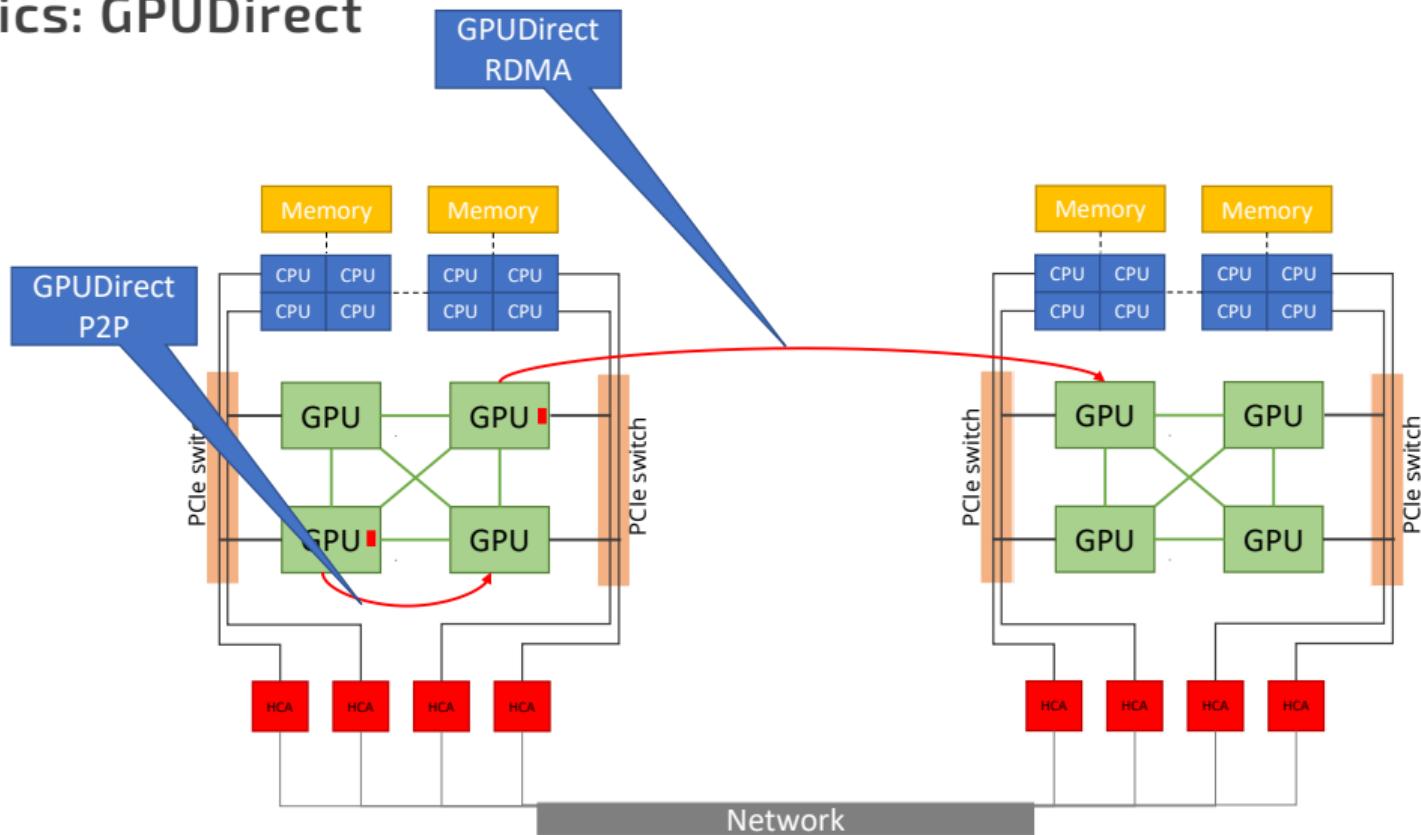
# Process Mapping on Multi GPU Systems

## One GPU per Process



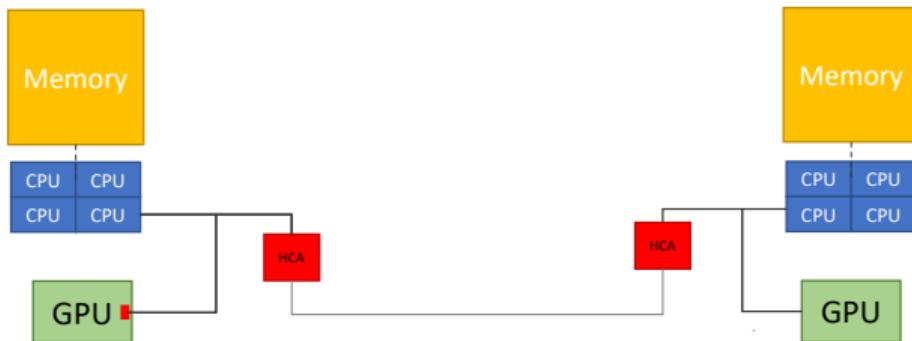
# Basics: GPUDirect

Slide quoted





# CUDA-aware MPI with 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, &stat);
```

# Summary

## 4L: Performance/Debugging Tools

# GPU Metrics in Nsight Systems

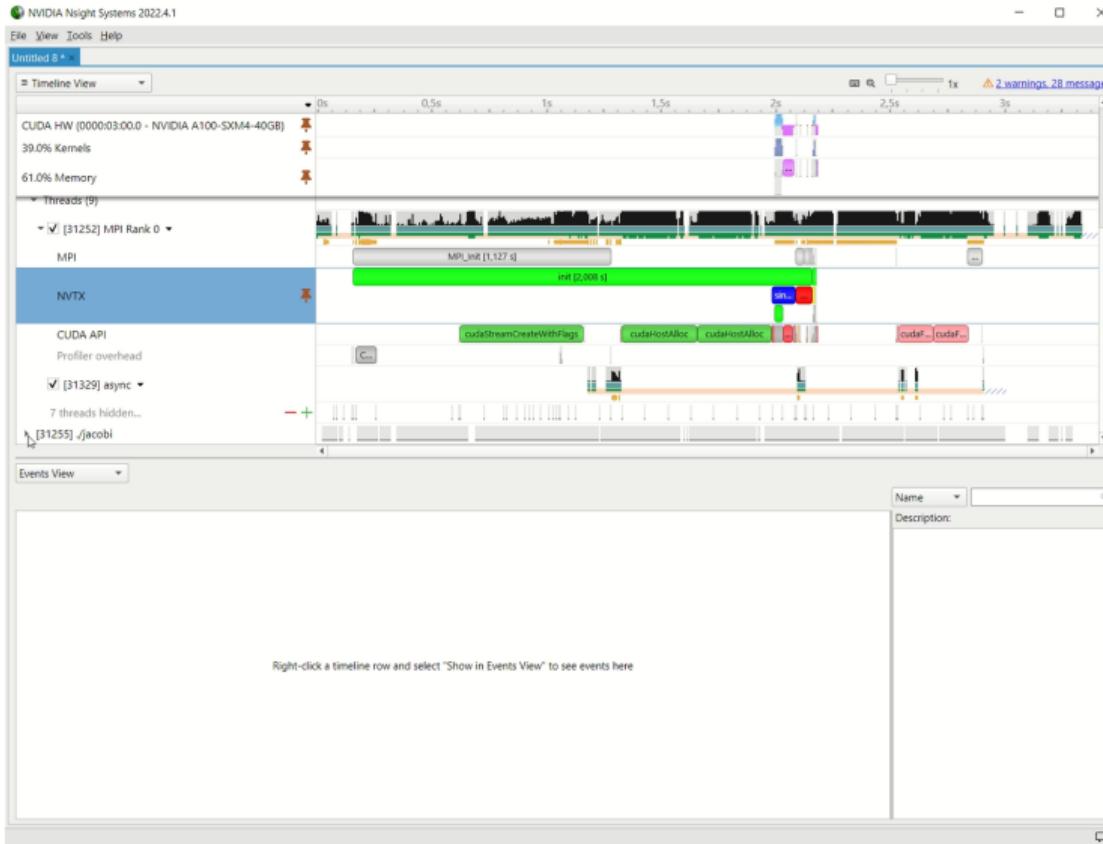
...and other traces you can activate

- Valuable low-overhead insight into HW usage:
  - SM instructions
  - DRAM Bandwidth, PCIe Bandwidth (GPUDirect)
- Also: Memory usage, Page Faults (higher overhead)
  - CUDA Programming guide: [Unified Memory Programming](#)
- Can save kernel-level profiling effort!
- nsys profile
 

```
--gpu-metrics-device=0
--cuda-memory-usage=true
--cuda-um-cpu-page-faults=true
--cuda-um-gpu-page-faults=true
./app
```

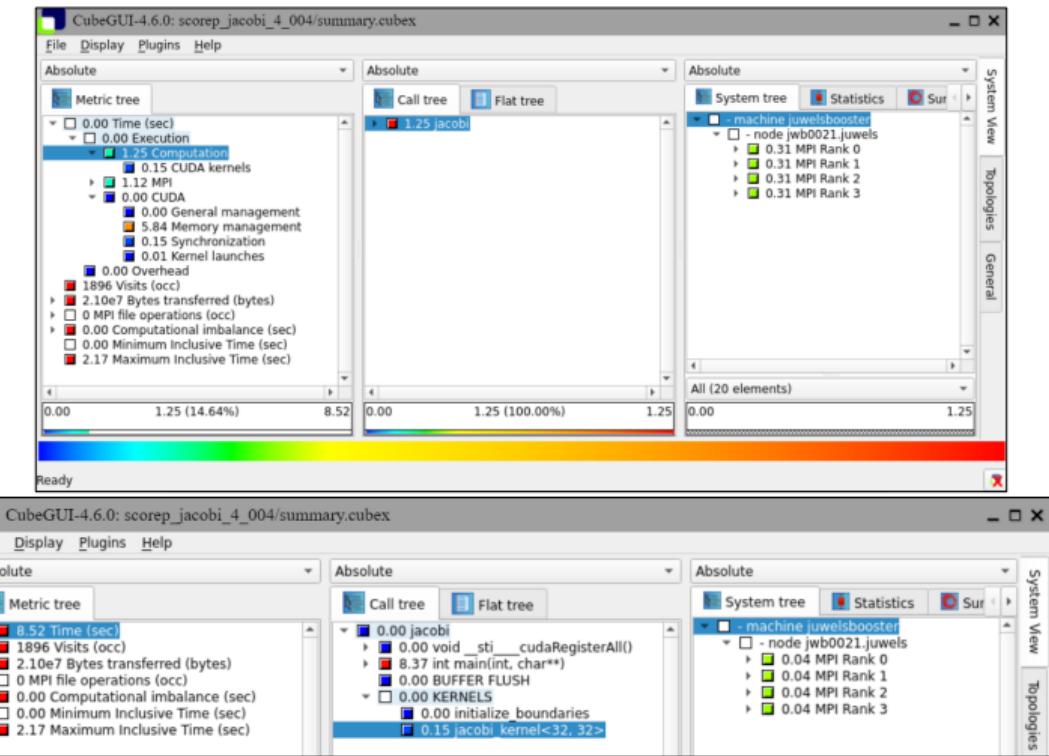


# Using Multiple Reports in Nsight Systems



# Scalasca / CUBE

- Breakdown of different metrics across functions and processes
- Left-to-right: Selection influences breakdown
- Expanding changes inclusive/exclusive
- Example analysis:
  - Detect computational imbalance
- <https://scalasca.org/>

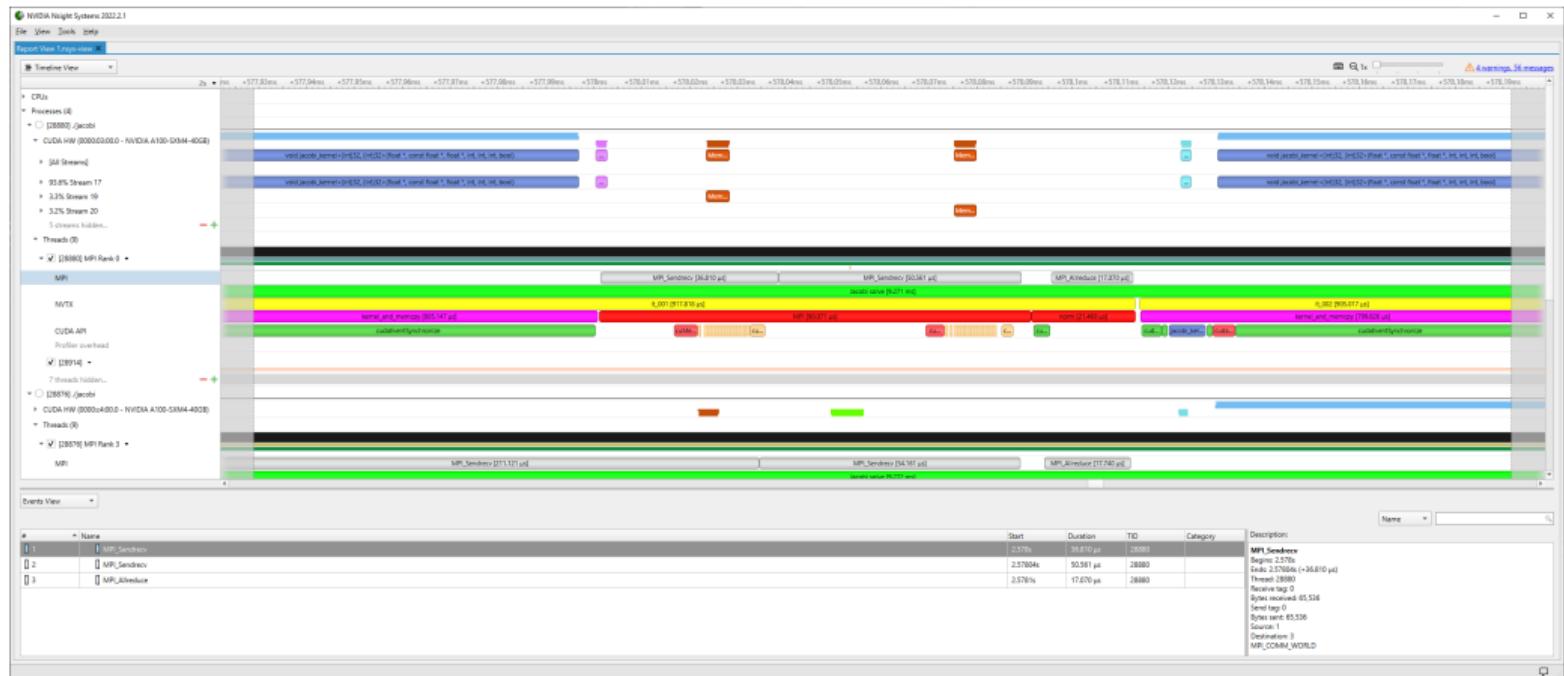


# Summary

## *5L: Optimization Techniques*

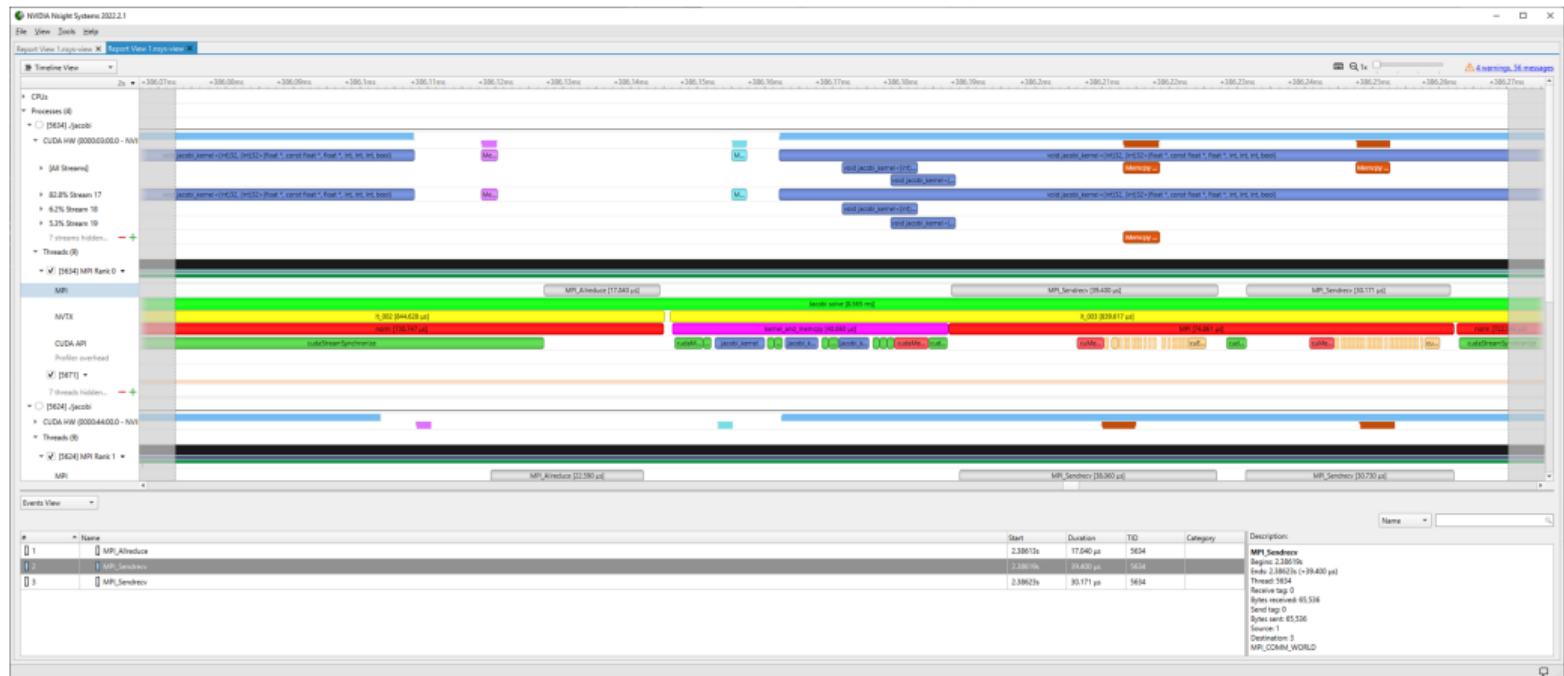
# Multi GPU Jacobi Nsight Systems Timeline

MPI 8 NVIDIA A100 40GB on JUWELS Booster



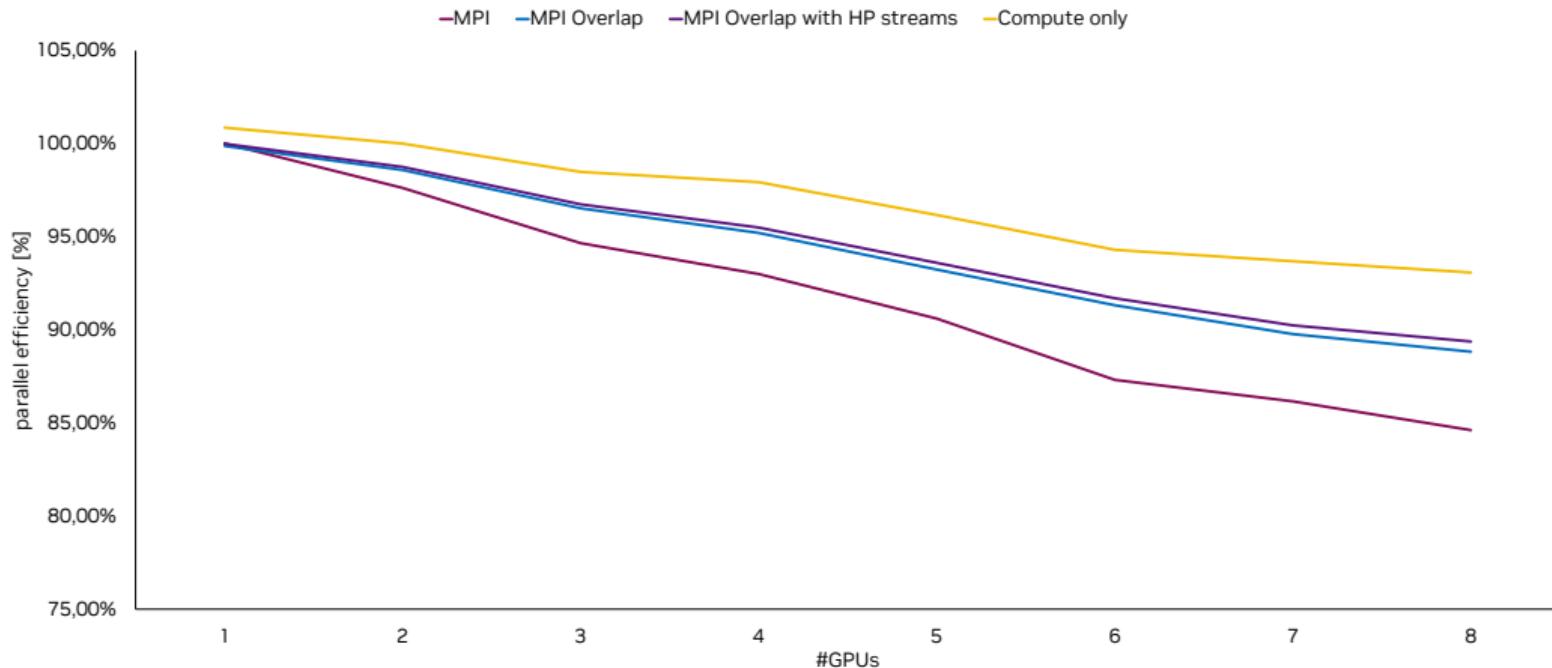
# Multi GPU Jacobi Nsight Systems Timeline

MPI Overlap 8 NVIDIA A100 40GB on JUWELS Booster



# Communication + Computation Overlap

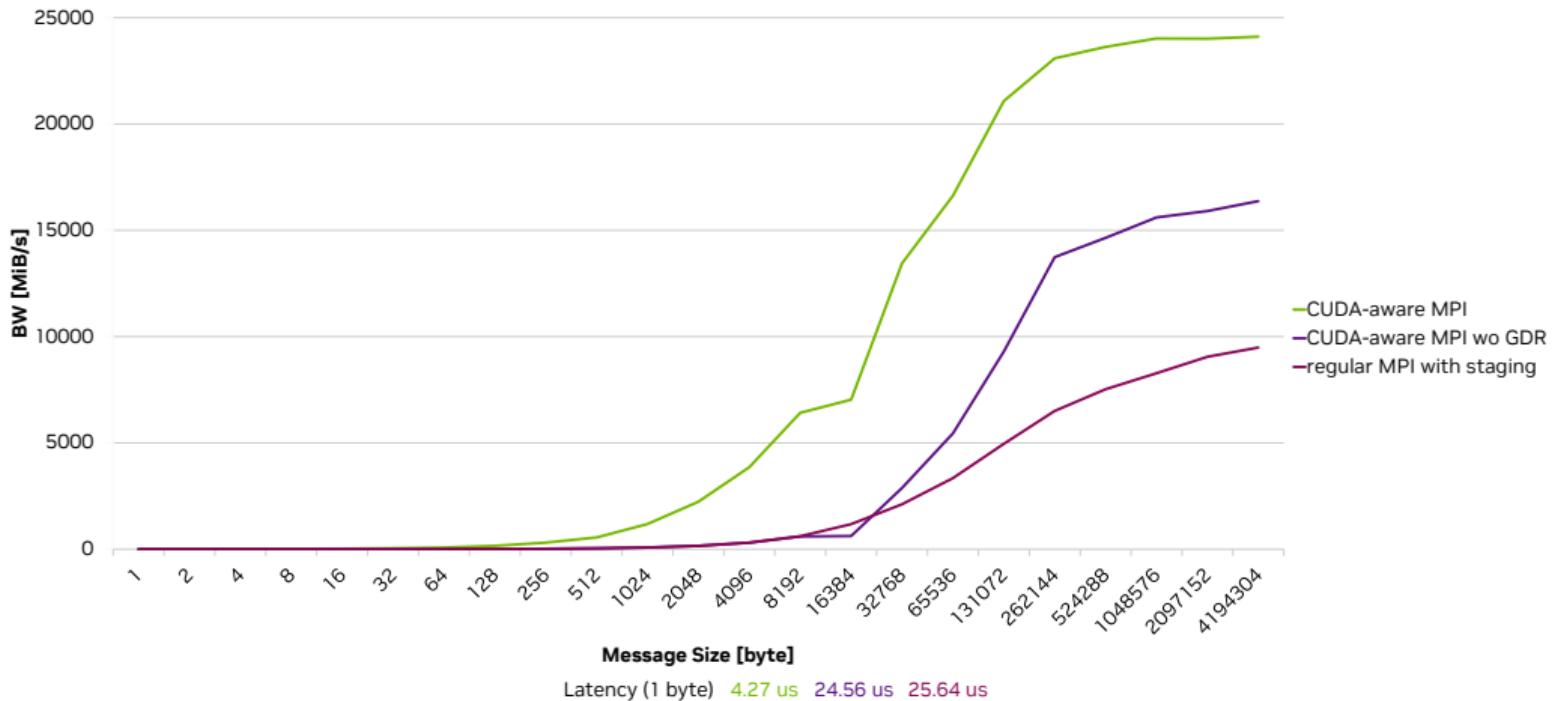
ParaStationMPI 5.4.10-1 – JUWELS Booster – NVIDIA A100 40 GB – Jacobi on 17408x17408



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

# Performance Results GPUDirect RDMA

Open MPI 4.1.0RC1 + UCX 1.9.0 on JUWELS Booster



# Summary

## *7L: NCCL, NVSHMEM*

## Communication Calls

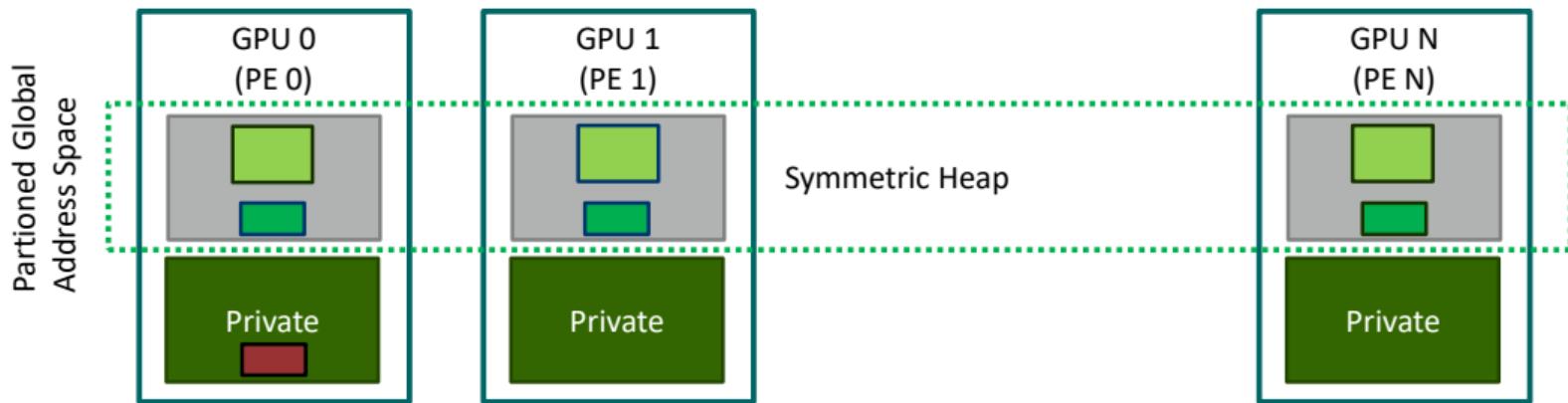
- Send/Recv

Supported  
for NCCL  
2.8+

```
ncclSend(void* sbuf, size_t count, ncclDataType_t type, int peer, ncclComm_t comm, cudaStream_t stream);  
ncclRecv(void* rbuf, size_t count, ncclDataType_t type, int peer, ncclComm_t comm, cudaStream_t stream);
```

```
ncclAllReduce(void* sbuf, void* rbuf, size_t count, ncclDataType_t type, ncclRedOp_t op, ncclComm_t comm, cudaStream_t stream);  
ncclBroadcast(void* sbuf, void* rbuf, size_t count, ncclDataType_t type, int root, ncclComm_t comm, cudaStream_t stream);  
ncclReduce(void* sbuf, void* rbuf, size_t count, ncclDataType_t type, ncclRedOp_t op, int root, ncclComm_t comm, cudaStream_t stream);  
ncclReduceScatter(void* sbuf, void* rbuf, size_t count, ncclDataType_t type, ncclRedOp_t op, ncclComm_t comm, cudaStream_t stream);  
ncclAllGather(void* sbuf, void* rbuf, size_t count, ncclDataType_t type, ncclComm_t comm, cudaStream_t stream);
```

## NVSHMEM Symmetric Memory Model



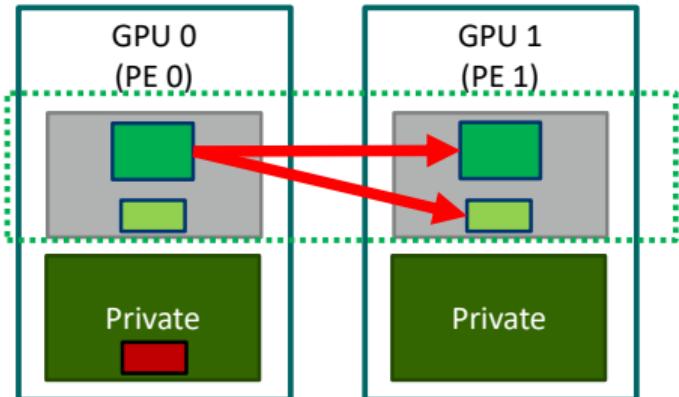
Symmetric objects are allocated collectively with the same size on every PE Symmetric memory:

`nvshmem_malloc( shared_size);`

Private memory: `cudaMalloc(...)`

Must be the  
same on all  
PEs

## NVSHMEM Host API Put



Copies  $nelems$  data elements of type  $T$  from symmetric objects  $src$  to  $dest$  on PE  $pe$

```
void nvshmem_<T>_put(T*dest, const T*source, size_t nelems, int pe);  
void nvshmem_x<T>_put_on_stream(T*dest, const T*src, size_t nelems, int pe, cudaStream_t stream);
```

The x marks extensions to  
the OpenSHMEM API

# Summary

**9L: Device-Initiated NVSHMEM**

## Asynchronous Task Graph

A Graph Node Is A CUDA Operation

*Sequence* of operations (nodes), connected by dependencies

Operations are one of:

Kernel Launch                    CUDA kernel running on GPU

CPU Function Call              Callback function on CPU

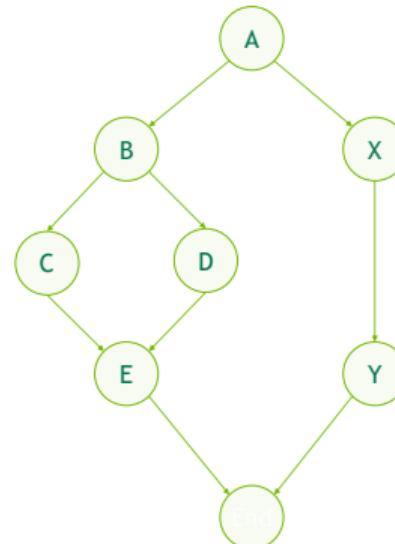
Memcpy/Memset                 GPU data management

Mem Alloc/Free                Memory management

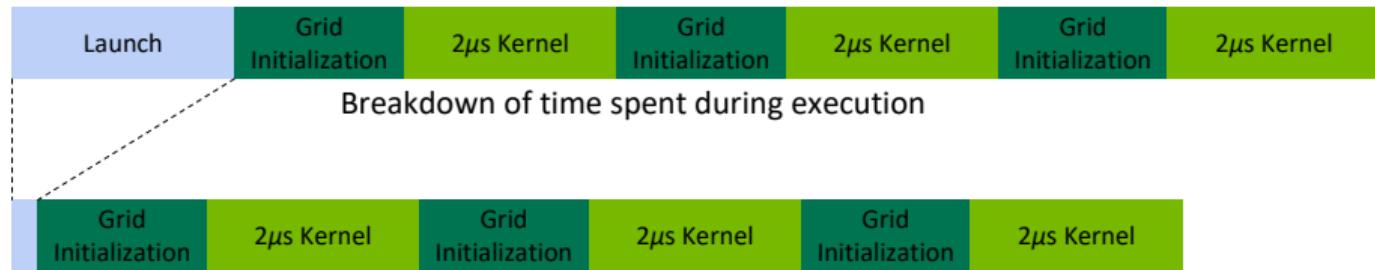
External Dependency           External semaphores/events

Sub-Graph                      Graphs are hierarchical

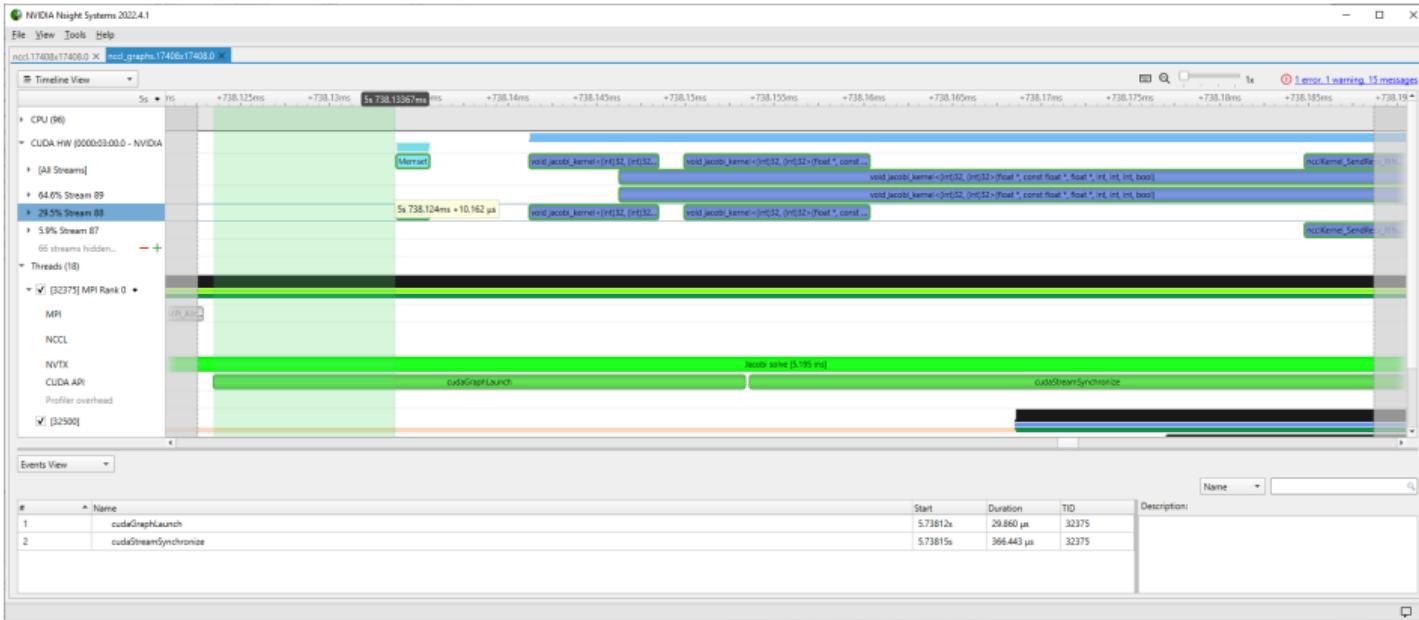
Nodes within a graph can also span multiple devices



## Where is performance coming from?



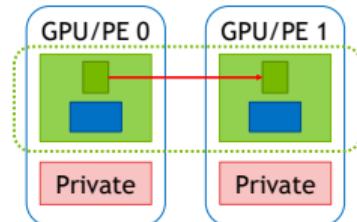
## Multi GPU Jacobi Nsight Systems Timeline



## NVSHMEM API

```
__device__ void nvshmemx_TYPENAME_put_nbi_block(TYPE *dest, const TYPE *source, size_t nelems, int pe)
```

- dest [OUT]: Symmetric address of the destination data object.
- source [IN]: Symmetric address of the object containing the data to be copied.
- nelems [IN]: Number of elements in the dest and source arrays.
- pe [IN]: The number of the remote PE.



Cooperative call: Needs to be called by all threads in a block. thread and warp are also available.

x in nvshmemx marks API as extension of the OpenSHMEM APIs.

[https://docs.nvidia.com/nvshmem/api/gen/api/rma.html#c.nvshmemx\\_TYPENAME\\_put\\_nbi\\_block](https://docs.nvidia.com/nvshmem/api/gen/api/rma.html#c.nvshmemx_TYPENAME_put_nbi_block)

TYPENAME can be: float, double, char, schar, short, int, long, longlong, uchar, ushort, uint, ulong, ulonglong, ..., ptrdiff

<https://docs.nvidia.com/nvshmem/api/gen/api/rma.html#stdrmatypes>

## Collective Kernel Launch

Ensures progress when using device-side inter-kernel synchronization

NVSHMEM Usage	CUDA Kernel launch
Device-Initiated Communication	Execution config syntax <<<...>>> or launch APIs
Device-Initiated Synchronization	<code>nvshmemx_collective_launch</code>

- CUDA's throughput computing model allows (encourages) grids much larger than a GPU can fit
- Inter-kernel synchronization requires producer and consumer threads to execute concurrently
- Collective launch guarantees co-residency using CUDA cooperative launch and requirement of 1PE/GPU

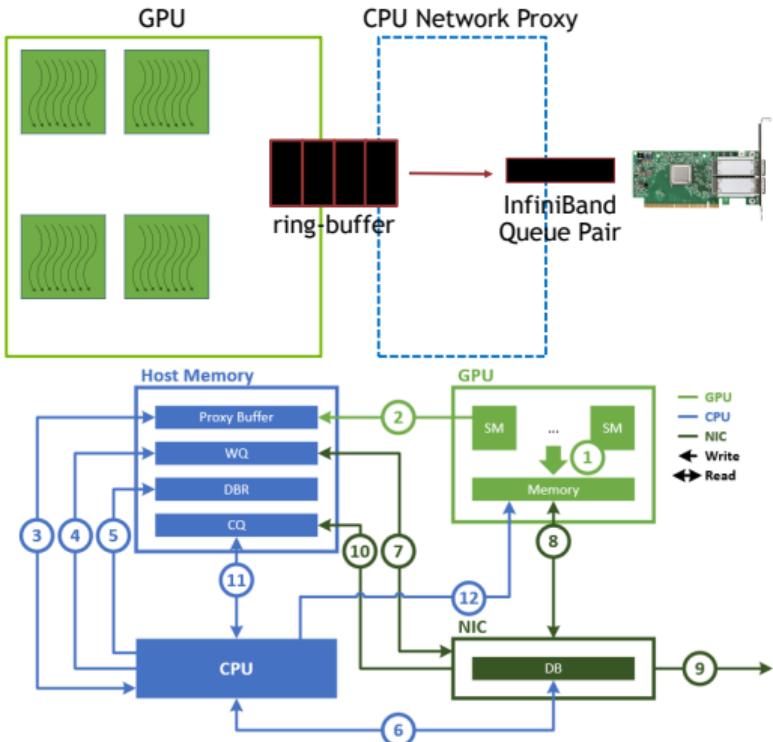
## Optimized Inter-Node Communication

NVSHMEM supports inter-node communication over InfiniBand, RoCE, and UCX (experimental)

Using GPUDirect RDMA (data plane)

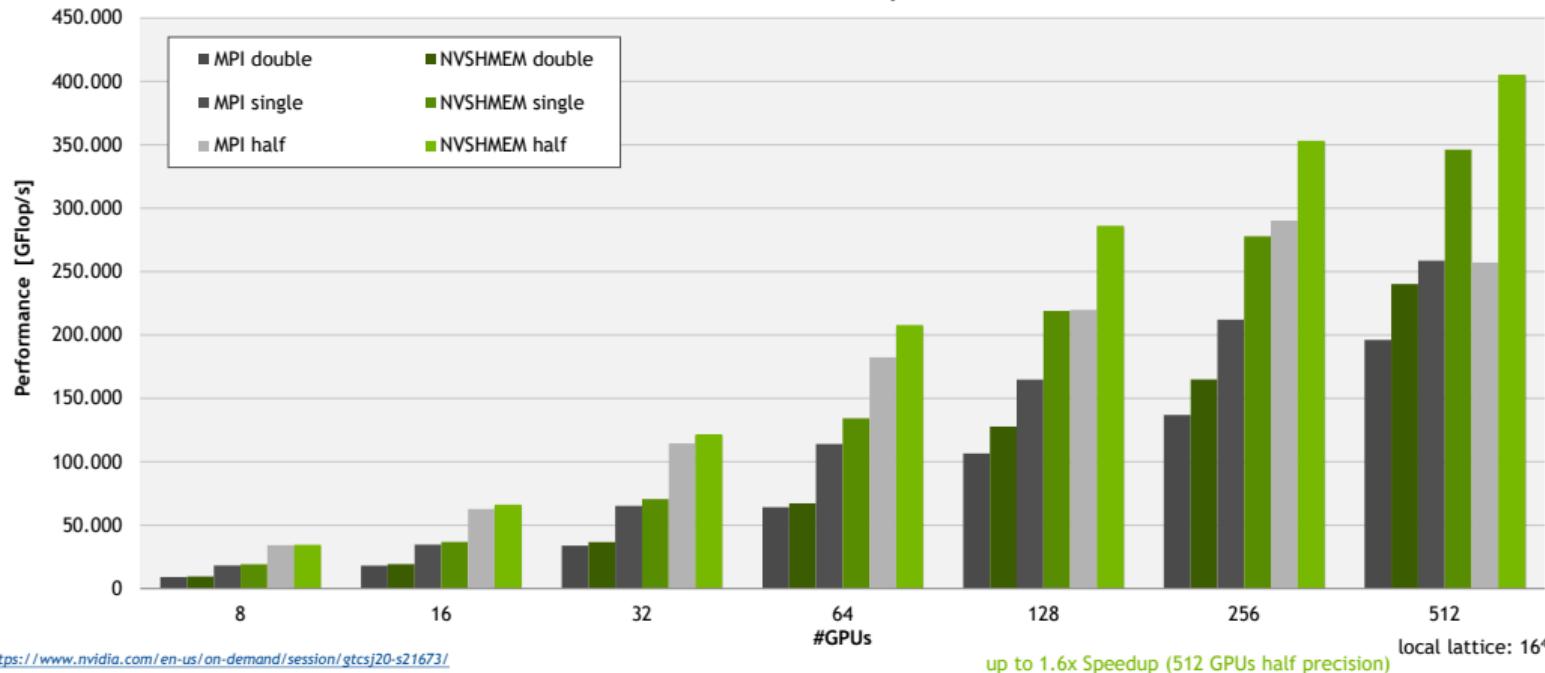
Reverse offloads network transfers from GPU to the CPU (control plane)

Ring buffer implementation avoids memory fences when interacting with CPU network proxy



## QUDA Strong Scaling on Selene

Lattice Quantum ChromoDynamics



*More: Other Languages/Models*

# OpenACC, OpenMP; Kokkos

- Directive-based GPU programming models work analogously to CUDA
- GPU-awareness via MPI configuration, no need to copyout or map(from)
- Using explicit device pointer necessary: host\_data use\_device / use\_device\_addr

```
#pragma acc host_data use_device( A )
MPI_Sendrecv( A+iy_start*nx+ix_start, (ix_end-ix_start), MPI_REAL_TYPE, top , 0,
              A+iy_end*nx+ix_start, (ix_end-ix_start), MPI_REAL_TYPE, bottom, 0,
              MPI_COMM_WORLD, MPI_STATUS_IGNORE );
}
```

- Advanced communication libraries can be used like any other library
- Kokkos similar: Use Kokkos::View and Kokkos::View::data() (see [Wiki](#))

```
Kokkos::View<double*> A("A", nx*ny);
MPI_Send(A.data(), int(A.size()), MPI_DOUBLE, bottom_rank, 0, COMM_WORLD);
```

# Python

- CUDA-awareness in MPI in Python available via  
`mpi4py`

# Python

- CUDA-awareness in MPI in Python available via  
`mpi4py`
- More Pythonic (and versatile): `cuNumeric`

# Python

- CUDA-awareness in MPI in Python available via  
`mpi4py`
- More Pythonic (and versatile): cuNumeric

```
import numpy as np
```

```
A = np.random.rand(N, N)
x = np.zeros(A.shape[1])
d = np.diag(A)
```

# Python

- CUDA-awareness in MPI in Python available via  
`mpi4py`
- More Pythonic (and versatile): `cuNumeric`

```
import cunumeric as np
```

```
A = np.random.rand(N, N)
x = np.zeros(A.shape[1])
d = np.diag(A)
```

# Python

- CUDA-awareness in MPI in Python available via  
mpi4py
- More Pythonic (and versatile): cuNumeric

```
import cunumeric as np
```

```
A = np.random.rand(N, N)
x = np.zeros(A.shape[1])
d = np.diag(A)
```

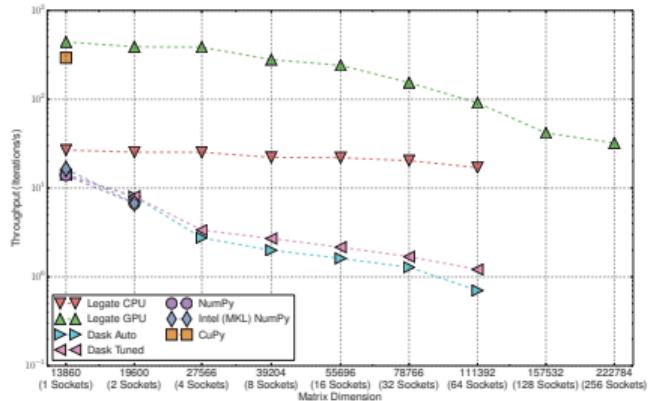
- cuNumeric [3]: transparently accelerates / distributes Numpy (and others)
  - Acceleration: Numpy kernel implementations for single-core CPU, multi-core CPU (OpenMP), and GPU (via libraries)
  - Distribution: OpenMP or MPI (via GASNet)
  - Type / size of task pool determined at start time via launcher script

# Python

- CUDA-awareness in MPI in Python available via `mpi4py`
- More Pythonic (and versatile): `cuNumeric`  
`import cunumeric as np`

```
A = np.random.rand(N, N)
x = np.zeros(A.shape[1])
d = np.diag(A)
```

- `cuNumeric` [3]: transparently accelerates / distributes Numpy (and others)
  - Acceleration: Numpy kernel implementations for single-core CPU, multi-core CPU (OpenMP), and GPU (via libraries)
  - Distribution: OpenMP or MPI (via GASNet)
  - Type / size of task pool determined at start time via launcher script



# Python

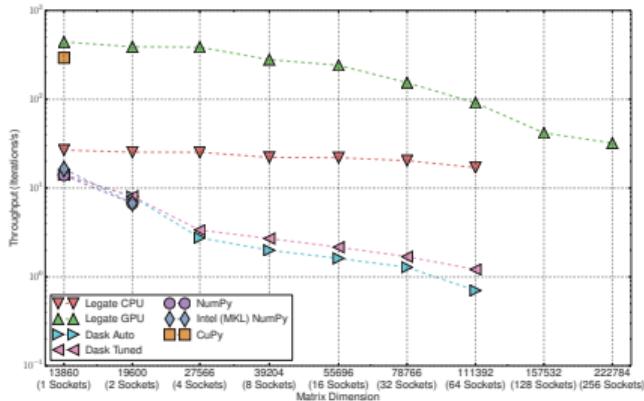
- CUDA-awareness in MPI in Python available via `mpi4py`
- More Pythonic (and versatile): `cuNumeric`  

```
import cunumeric as np
```

```
A = np.random.rand(N, N)
x = np.zeros(A.shape[1])
d = np.diag(A)
```

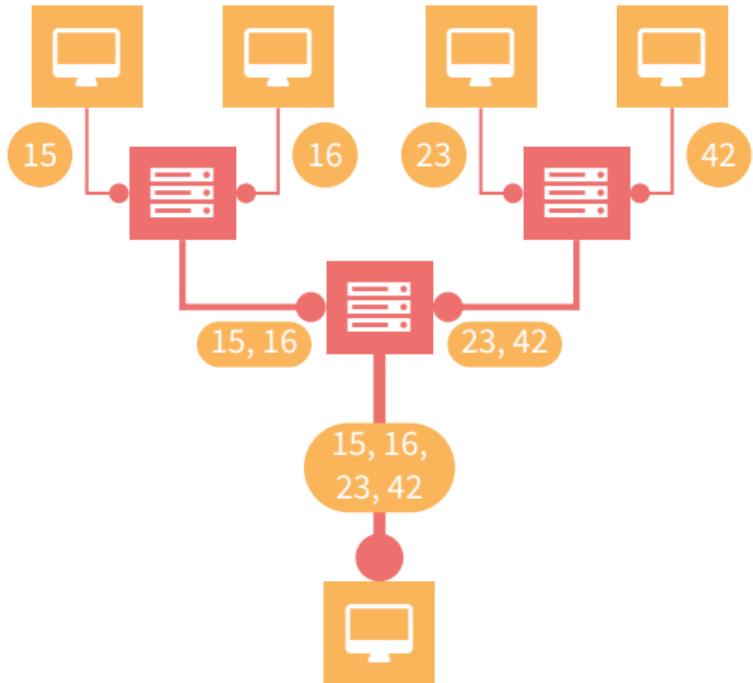
- `cuNumeric` [3]: transparently accelerates / distributes Numpy (and others)
  - Acceleration: Numpy kernel implementations for single-core CPU, multi-core CPU (OpenMP), and GPU (via libraries)
  - Distribution: OpenMP or MPI (via GASNet)
  - Type / size of task pool determined at start time via launcher script

→ <https://github.com/nv-legate/cunumeric/>



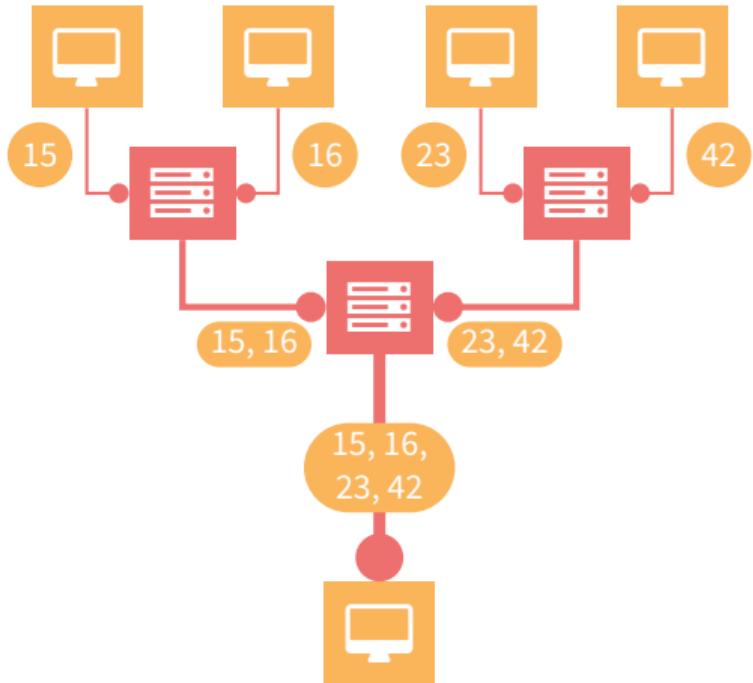
# *More: In-Network Computing*

# In-Network Computing

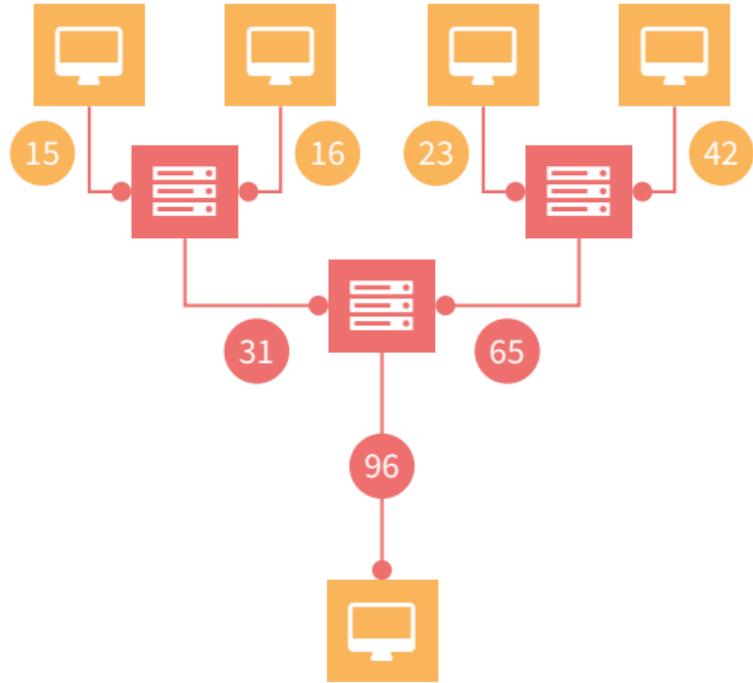


*Traditional Reduce()*

# In-Network Computing



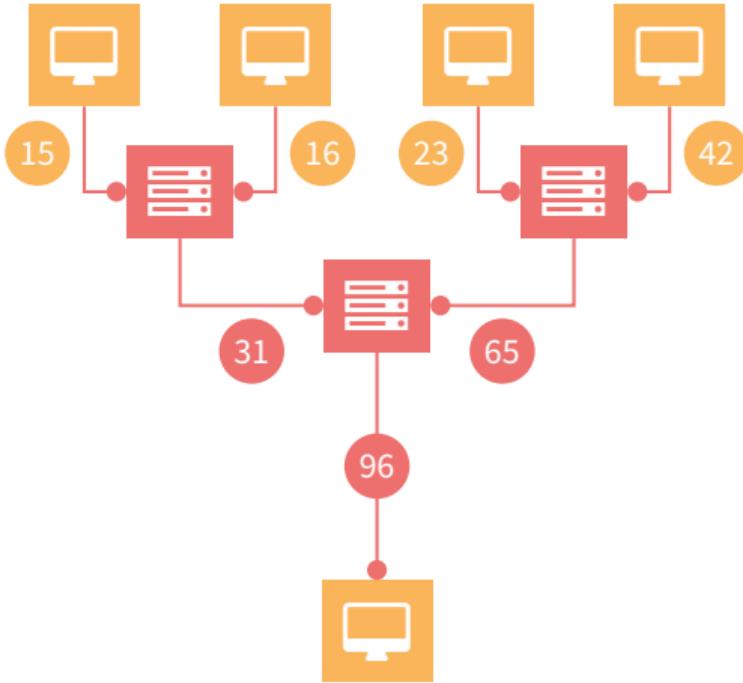
*Traditional Reduce()*



*Switch-supported Reduce()*

# In-Network Computing

- Usually, network devices (switches, HCAs) just forward to computing devices
- Modern hardware offers in-network computation
- Works also with GPUs
- Less latency, less traffic
- Especially for communication-intensive collectives like AllReduce()



*Switch-supported Reduce()*

# In-Network Computing Libraries

MPI

- MPI** MPI runtime transparently offloads specific collective operations to network, if enabled  
(OpenMPI, e.g. bundled in NVIDIA's HPC-X; MVAPICH2-X; also NCCL via plugin)

# In-Network Computing Libraries

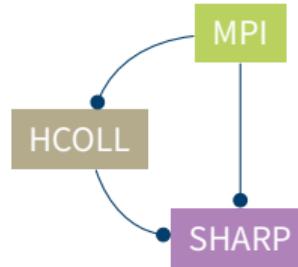
**MPI** MPI runtime transparently offloads specific collective operations to network, if enabled  
(OpenMPI, e.g. bundled in NVIDIA's HPC-X; MVAPICH2-X; also NCCL via plugin)



**SHARP** Lowest / base level API (*Scalable Hierarchical Aggregation and Reduction Protocol*)  
`libsharp_coll`: interface, `libsharp`: backend

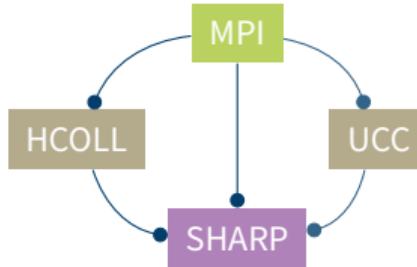
# In-Network Computing Libraries

- MPI** MPI runtime transparently offloads specific collective operations to network, if enabled  
(OpenMPI, e.g. bundled in NVIDIA's HPC-X; MVAPICH2-X; also NCCL via plugin)
- HCOLL** NVIDIA's HCOLL (*Hierarchical Collectives*; libhcoll), middleware, via OpenMPI / HPC-X
- SHARP** Lowest / base level API (*Scalable Hierarchical Aggregation and Reduction Protocol*)  
`libsharp_coll`: interface, `libsharp`: backend



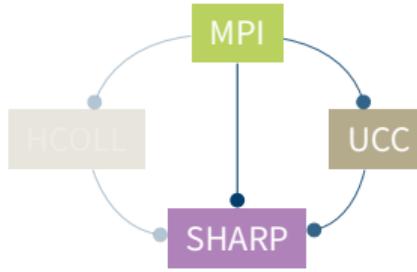
# In-Network Computing Libraries

- MPI** MPI runtime transparently offloads specific collective operations to network, if enabled  
(OpenMPI, e.g. bundled in NVIDIA's HPC-X; MVAPICH2-X; also NCCL via plugin)
- HCOLL** NVIDIA's HCOLL (*Hierarchical Collectives*; `libhcoll`), middleware, via OpenMPI / HPC-X
- UCC** New intermediate layer (from UCF initiative (UCX)) for *Unified Collective Communication*, alternative to HCOLL  
→ <https://github.com/openucx/ucc>
- SHARP** Lowest / base level API (*Scalable Hierarchical Aggregation and Reduction Protocol*)  
`libsharp_coll`: interface, `libsharp`: backend



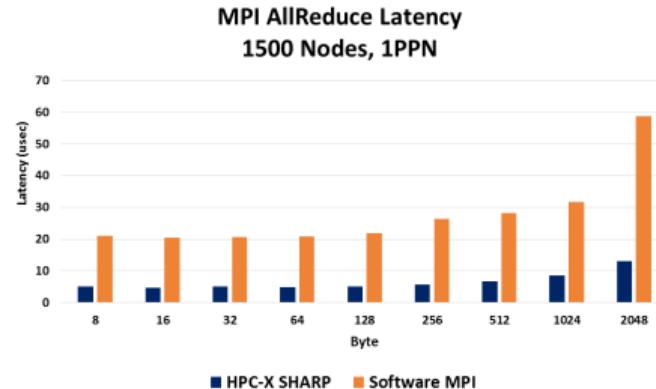
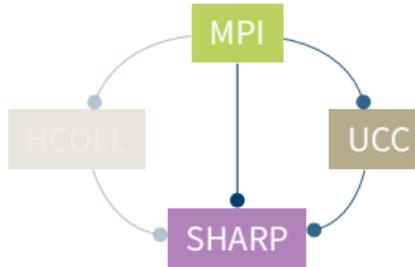
# In-Network Computing Libraries

- MPI** MPI runtime transparently offloads specific collective operations to network, if enabled  
(OpenMPI, e.g. bundled in NVIDIA's HPC-X; MVAPICH2-X; also NCCL via plugin)
- HCOLL** NVIDIA's HCOLL (*Hierarchical Collectives*; libhcoll), middleware, via OpenMPI / HPC-X
- UCC** New intermediate layer (from UCF initiative (UCX)) for *Unified Collective Communication*, alternative to HCOLL  
→ <https://github.com/openucx/ucc>
- SHARP** Lowest / base level API (*Scalable Hierarchical Aggregation and Reduction Protocol*)  
`libsharp_coll`: interface, `libsharp`: backend



# In-Network Computing Libraries

- MPI** MPI runtime transparently offloads specific collective operations to network, if enabled  
(OpenMPI, e.g. bundled in NVIDIA's HPC-X; MVAPICH2-X; also NCCL via plugin)
- HCOLL** NVIDIA's HCOLL (*Hierarchical Collectives*; libhcoll), middleware, via OpenMPI / HPC-X
- UCC** New intermediate layer (from UCF initiative (UCX)) for *Unified Collective Communication*, alternative to HCOLL  
→ <https://github.com/openuxc/ucc>
- SHARP** Lowest / base level API (*Scalable Hierarchical Aggregation and Reduction Protocol*)  
`libsharp_coll`: interface, `libsharp`: backend



Graph by Gil Bloch / Mellanox (2019)[11]

# Other Vendors

# AMD

- AMD Instinct GPUs entered HPC with a boom
- Multi-node ecosystem maturing rapidly
- Key technology already developed, mimicking NVIDIA's strategy
- UCX is ROCm enabled ([how-to ↗](#)); MVAPICH2-GDR [12] also optimized

Technology	NVIDIA	AMD
RDMA Support	GPUDirect RDMA	ROCmRDMA
Peer to Peer	GPUDirect P2P	ROCm IPC
Direct CPU Access (PCIe BAR)	GDRCopy BAR1	LargeBar
Accelerated Collectives	NCCL	RCCL
OpenSHMEM	NVSHMEM	ROC_SHMEM

# AMD HIP Jacobi MPI Example

- Procedure: hipify-perl → fix errors → compile

- Code example

```
hipGetDeviceCount(&num_devices);
hipSetDevice(local_rank%num_devices);
real* a_ref_h;
hipHostMalloc(&a_ref_h, nx * ny * sizeof(real));
```

- Compilation example

```
HIP_PLATFORM=amd hipcc --offload-arch=gfx90a -std=c++14 -munsafe-fp-atomics -O3 -fopenmp
→ -I${MPI_HOME}/include -c -o jacobi.cu.hip.o jacobi.cu.hip
```

```
HIP_PLATFORM=amd hipcc --offload-arch=gfx90a -std=c++14 -munsafe-fp-atomics -O3
→ -I${MPI_HOME}/include -L${MPI_HOME}/lib -lmpi --gcc-toolchain=${EBROOTGCCCORE} -o
→ jacobi.amd jacobi.cu.hip.o
```

- Needed: ROCm-aware UCX (`UCX_TLS=rc_x,self,sm,rocm_copy,rocm_ipc`)

# Summary, Conclusion

# Summary, Conclusion

Efficient multi-node GPU computing is efficient multi-node computing with least possible amount of CPU

- Many advanced technologies and techniques in place to enable large-scale GPU applications
- GPU-aware MPI is key enabler
- On top / orthogonal: NCCL, NVSHMEM, ...
- Profiling important to pinpoint bottlenecks (*in HPC, bad performance is a bug*)
- Supercomputer of tutorial: JUWELS Booster, European flagship system based on A100 GPUs and HDR200 InfiniBand network
- Tutorial with team experienced in distributed GPU workloads
- Appendix: *Links, references*

# Summary, Conclusion

Efficient multi-node GPU computing is efficient multi-node computing with least possible amount of CPU

- Many advanced technologies and techniques in place to enable large-scale GPU applications
- GPU-aware MPI is key enabler
- On top / orthogonal: NCCL, NVSHMEM, ...
- Profiling important to pinpoint bottlenecks (*in HPC, bad performance is often based on A100*)
- Supercomputer of tutorial: JUWELS Booster, European flagship system based on A100 GPUs and HDR200 InfiniBand network
- Tutorial with team experienced in distributed GPU workloads
- Appendix: *Links, references*

Thank you  
for your attention!  
[a.herten@fz-juelich.de](mailto:a.herten@fz-juelich.de)

# Appendix

## References

# Links I

- [1] *JUWELS Booster Overview.* URL:  
<https://apps.fz-juelich.de/jsc/hps/juwels/booster-overview.html>.
- [2] *Support of GPU-aware MPI in mpi4py.* URL:  
<https://mpi4py.readthedocs.io/en/stable/overview.html#support-for-gpu-aware-mpi>.
- [4] *Legate (Numpy).* URL: <https://github.com/nv-legate/legate.numpy>.
- [5] *NVIDIA: HPC-X.* URL: <https://docs.mellanox.com/category/hpcx>.
- [6] *MVAPICH2.* URL: <https://mvapich.cse.ohio-state.edu/>.
- [7] *NVIDIA: NCCL SHARP Plugin.* URL:  
<https://github.com/Mellanox/nccl-rdma-sharp-plugins>.

# Links II

- [8] NVIDIA: HCOLL (via HPC-X). URL:  
<https://docs.mellanox.com/display/HPCXv29/HCOLL>.
- [9] Unified Communication Framework (UCF) Consortium. URL:  
<https://ucfconsortium.org/>.
- [10] NVIDIA: SHARP. URL: <https://docs.mellanox.com/category/mlnxsharp>.

# References I

- [3] Michael Bauer and Michael Garland. “Legate NumPy: Accelerated and Distributed Array Computing.” In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC ’19. Denver, Colorado: Association for Computing Machinery, 2019. ISBN: 9781450362290. doi: [10.1145/3295500.3356175](https://doi.org/10.1145/3295500.3356175). URL: <https://doi.org/10.1145/3295500.3356175> (pages 34–40).
- [11] Gil Bloch. “SHARP Tutorial.” In: *HPC Advisory Council 2019 Lugano Workshop*. 2019. URL: [http://www.hpcadvisorycouncil.com/events/2019/swiss-workshop/pdf/020419/G\\_Bloch\\_Mellanox\\_SHARP\\_02042019.pdf](http://www.hpcadvisorycouncil.com/events/2019/swiss-workshop/pdf/020419/G_Bloch_Mellanox_SHARP_02042019.pdf) (pages 45–50).

## References II

- [12] Kawthar Shafie Khorassani et al. “Designing a ROCm-Aware MPI Library for AMD GPUs: Early Experiences.” In: *High Performance Computing*. Ed. by Bradford L. Chamberlain et al. Cham: Springer International Publishing, 2021, pp. 118–136. ISBN: 978-3-030-78713-4. URL: [https://link.springer.com/chapter/10.1007%2F978-3-030-78713-4\\_7](https://link.springer.com/chapter/10.1007%2F978-3-030-78713-4_7) (page 52).