GPU COMPUTING
2 – MULTI GPU

Mathias Wagner, Lattice Practices 2024

MOTIVATION

Why use multiple GPUs?

Need to compute larger, e.g. bigger networks, car models, …

Need to compute faster, e.g. weather prediction

Better energy efficiency with dense nodes with multiple GPUs

EXAMPLE: JACOBI SOLVER

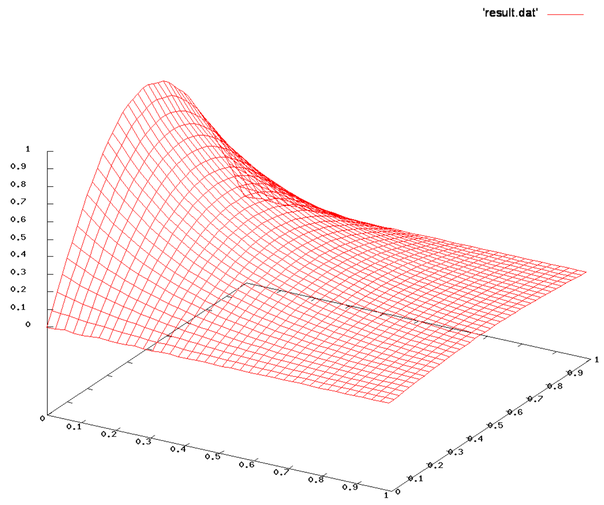
Solves the 2D-Laplace Equation on a rectangle

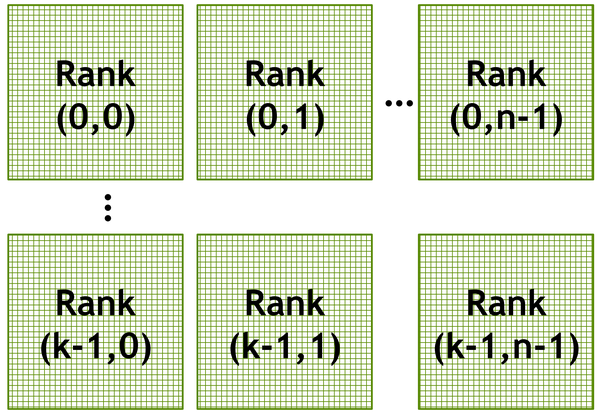
$$\Delta \boldsymbol{u}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{0} \; \forall \; (\boldsymbol{x}, \boldsymbol{y}) \in \Omega \backslash \boldsymbol{\delta}\Omega$$

Dirichlet boundary conditions (constant values on boundaries)

$$\boldsymbol{u}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}) \; \forall \; (\boldsymbol{x}, \boldsymbol{y}) \in \boldsymbol{\delta}\Omega$$

2D domain decomposition with n x k domains





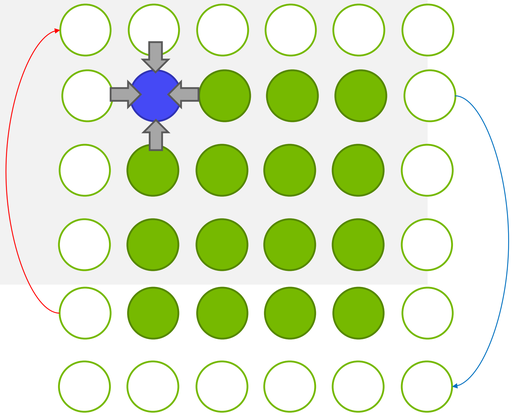
EXAMPLE: JACOBI SOLVER

Single GPU

While not converged

Do Jacobi step:

```
for( int iy = 1; iy < ny-1; iy++ )
for( int ix = 1; ix < nx-1; ix++ )
    a_new[iy*nx+ix] = -0.25 *
                      -( a[ iy   *nx+(ix+1)] + a[ iy   *nx+ix-1]
                       + a[(iy-1)*nx+ ix   ] + a[(iy+1)*nx+ix  ] );
```



Apply periodic boundary conditions

Swap `a_new` and `a`

Next iteration

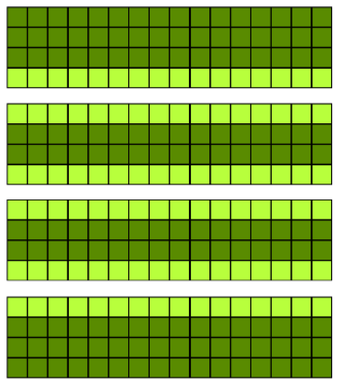
DOMAIN DECOMPOSITION

Different Ways to split the work between processes:

Minimize number of neighbors:

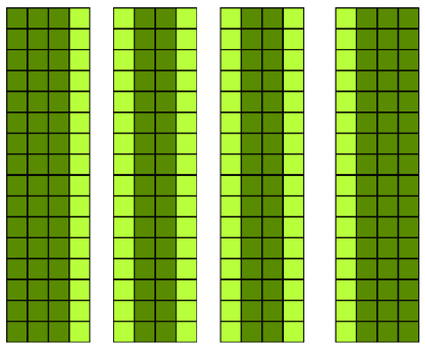
Communicate to less neighbors

Optimal for latency bound communication



Horizontal Stripes

Contiguous if data is row-major



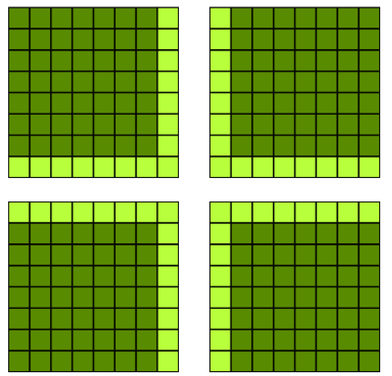
Vertical Stripes

Contiguous if data is column-major

Minimize surface area/volume ratio:

Communicate less data

Optimal for bandwidth bound communication



Tiles

EXAMPLE: JACOBI SOLVER

Multi GPU

While not converged

Do Jacobi step:

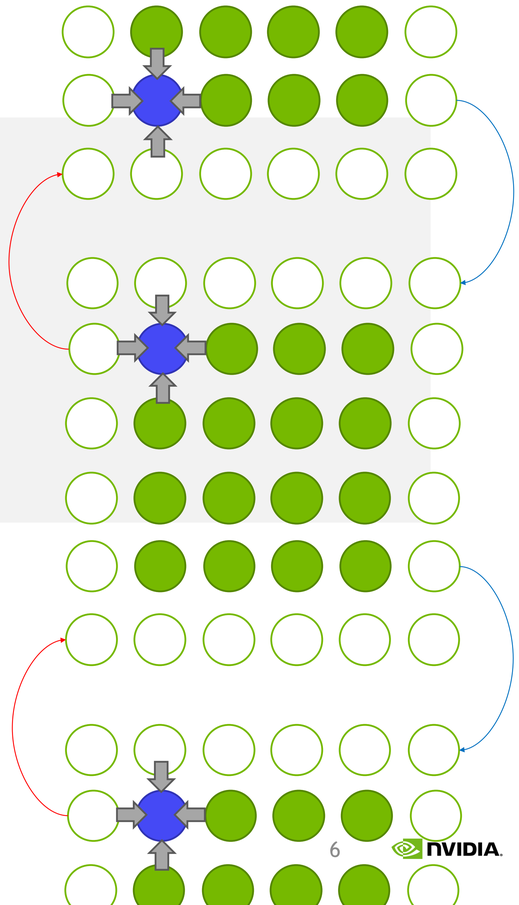
```
for( int iy = iy_start; iy < iy_end; iy++ )
for( int ix = 1;          ix < nx-1;   ix++ )
    a_new[iy*nx+ix] = -0.25 *
                      -( a[ iy   *nx+(ix+1)] + a[ iy   *nx+ix-1]
                       + a[(iy-1)*nx+ ix   ] + a[(iy+1)*nx+ix  ] );
```

Apply periodic boundary conditions

Halo exchange

One-step with ring exchange

Swap `a_new` and `a` and start next iteration



MESSAGE PASSING INTERFACE - MPI

Standard to exchange data between processes via messages

- Defines API to exchanges messages
  - Point to Point: e.g. `MPI_Send`, `MPI_Recv`
  - Collectives: e.g. `MPI_Reduce`

Multiple implementations (open source and commercial)

- Bindings for C/C++, Fortran, Python, …
- E.g. MPICH, OpenMPI, MVAPICH, IBM Platform MPI, Cray MPT, …

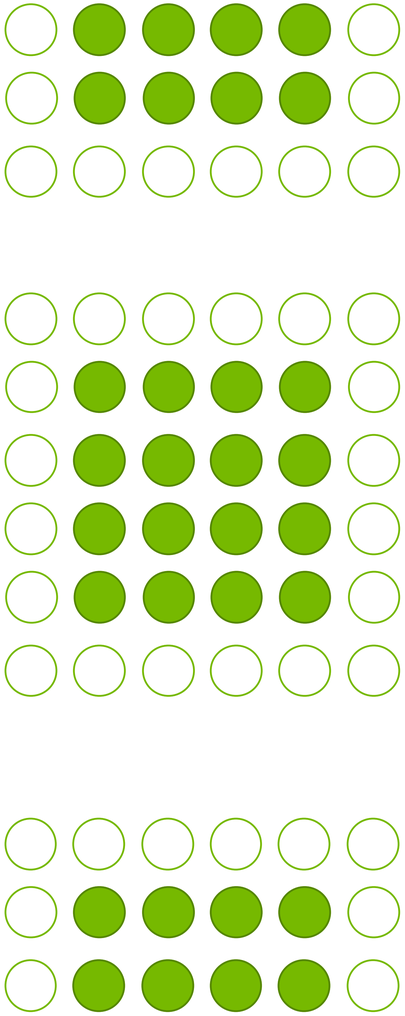
MPI - SKELETON

```c
#include <mpi.h>
int main(int argc, char *argv[]) {
    int rank,size;
    /* Initialize the MPI library */
    MPI_Init(&argc,&argv);
    /* Determine the calling process rank and total number of ranks */
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    MPI_Comm_size(MPI_COMM_WORLD,&size);
    /* Call MPI routines like MPI_Send, MPI_Recv, ... */
    ...
    /* Shutdown MPI library */
    MPI_Finalize();
    return 0;
}
```

EXAMPLE JACOBI

Top/Bottom Halo

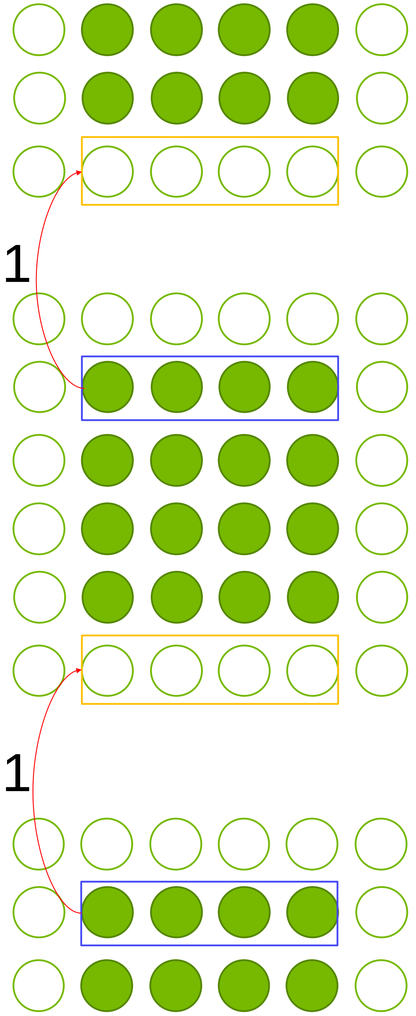
```
MPI_Sendrecv(a_new+iy_start*nx, nx, MPI_FLOAT, top , 0,
             a_new+(iy_end*nx), nx, MPI_FLOAT, bottom, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```



EXAMPLE JACOBI

Top/Bottom Halo

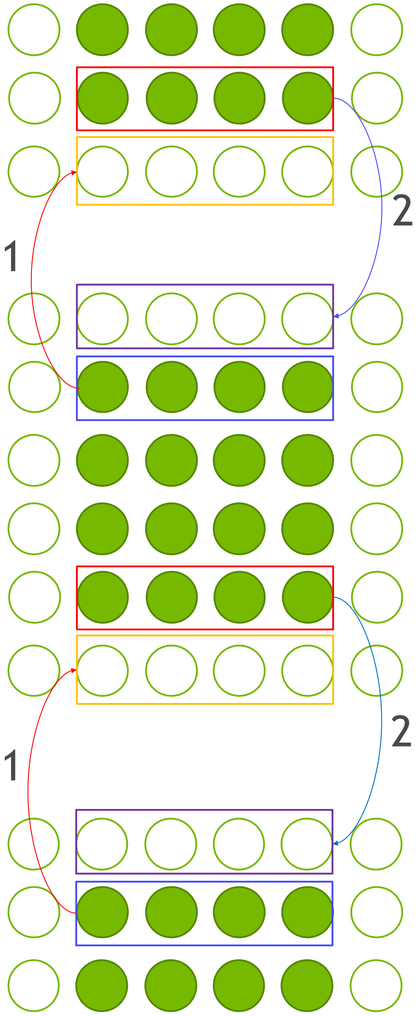
```
MPI_Sendrecv(a_new+iy_start*nx, nx, MPI_FLOAT, top , 0,
             a_new+(iy_end*nx), nx, MPI_FLOAT, bottom, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```



EXAMPLE JACOBI

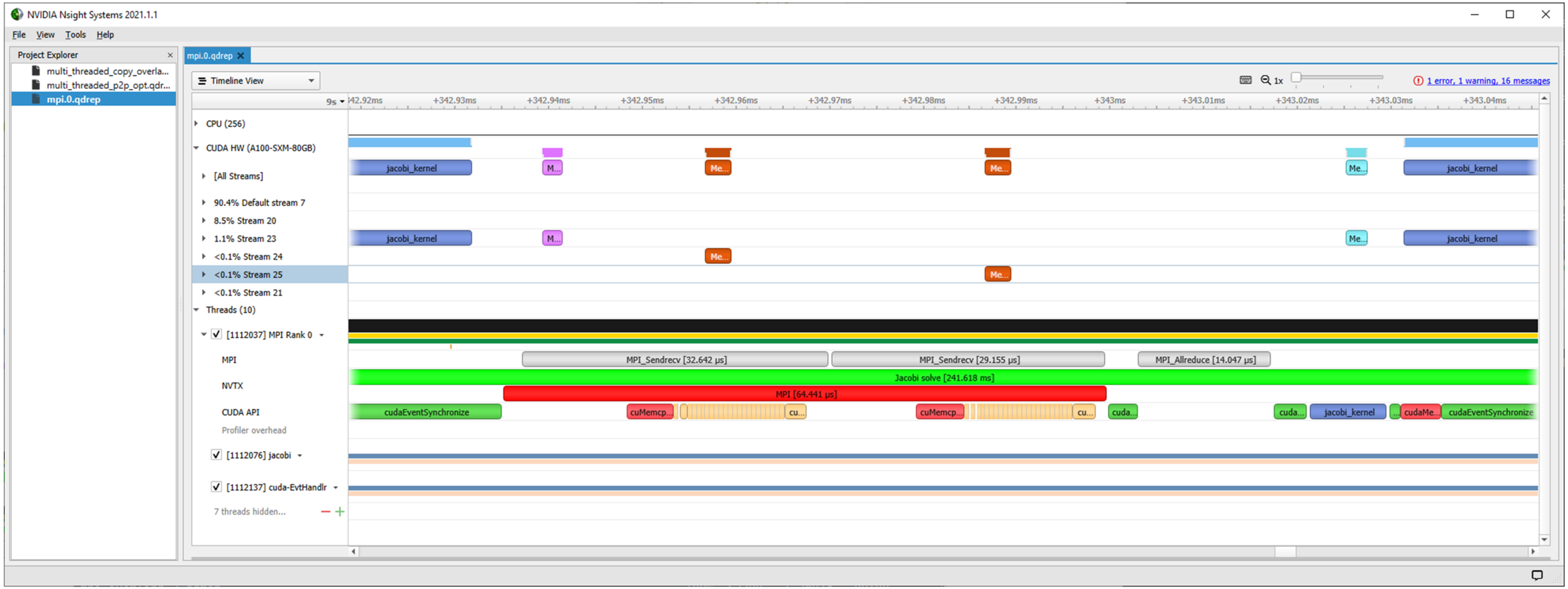
Top/Bottom Halo

```
MPI_Sendrecv(a_new+iy_start*nx, nx, MPI_FLOAT, top , 0,
             a_new+(iy_end*nx), nx, MPI_FLOAT, bottom, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);

MPI_Sendrecv(a_new+(iy_end-1)*nx, nx, MPI_FLOAT, bottom, 0,
             a_new, nx, MPI_FLOAT, top, 0, MPI_COMM_WORLD,
             MPI_STATUS_IGNORE);
```



MULTI GPU JACOBI NSIGHT SYSTEMS TIMELINE

MPI 8 NVIDIA A100 80GB on DGX A100

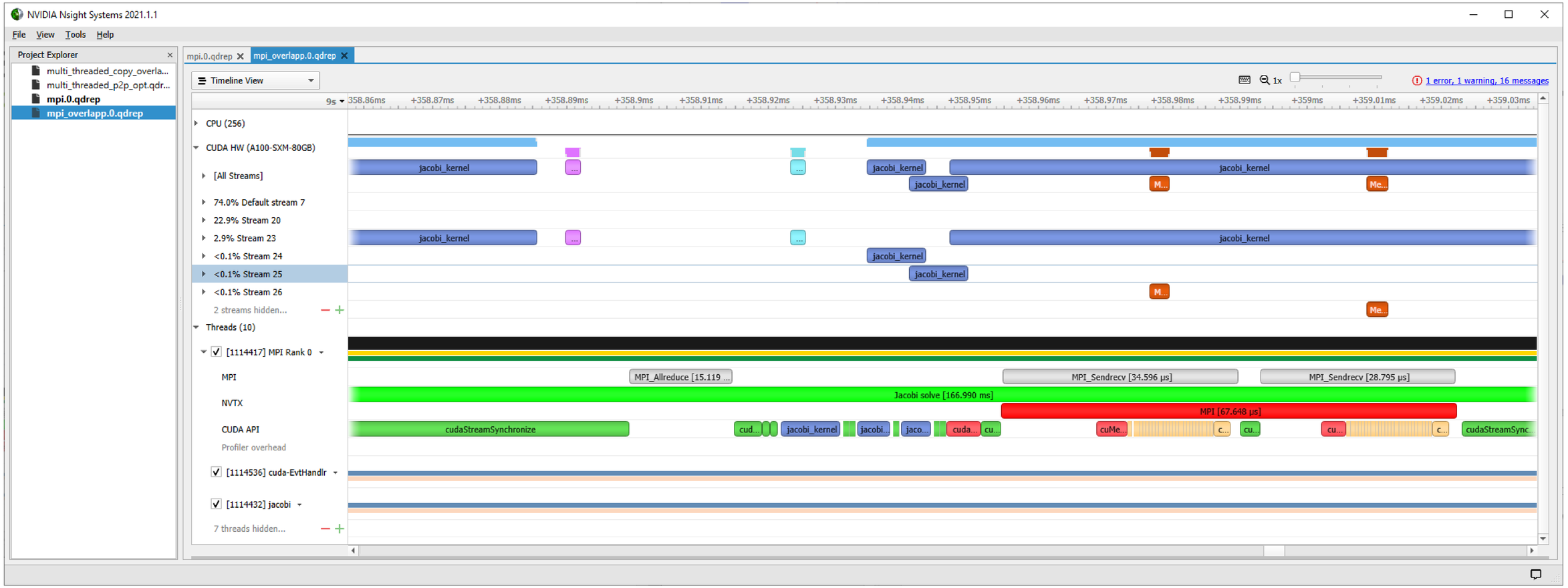


MPI COMMUNICATION + COMPUTATION OVERLAP

```
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;
cudaStreamSynchronize( push_top_stream );
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 );
cudaStreamSynchronize( push_bottom_stream );
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

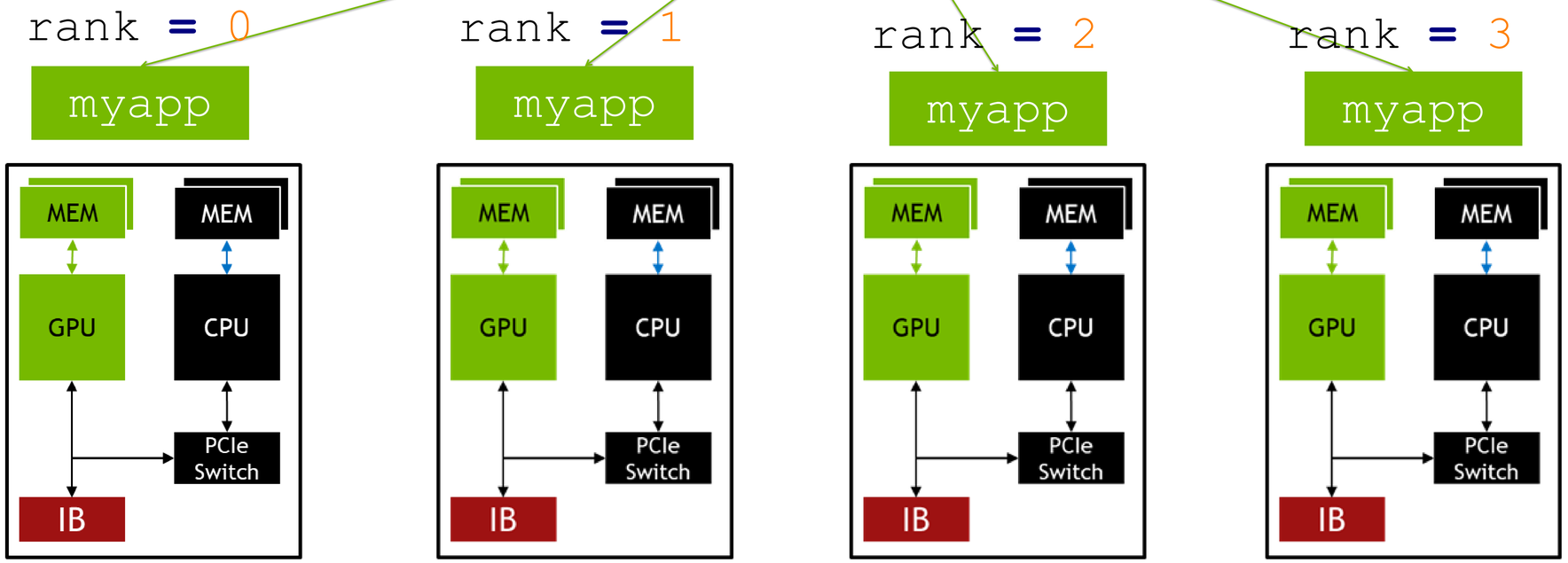
MPI Overlap 8 NVIDIA A100 80GB on DGX A100



NVIDIA

MPI
Compiling and Launching

```
$ mpicc -o myapp myapp.c
$ mpirun -np 4 ./myapp <args>
```

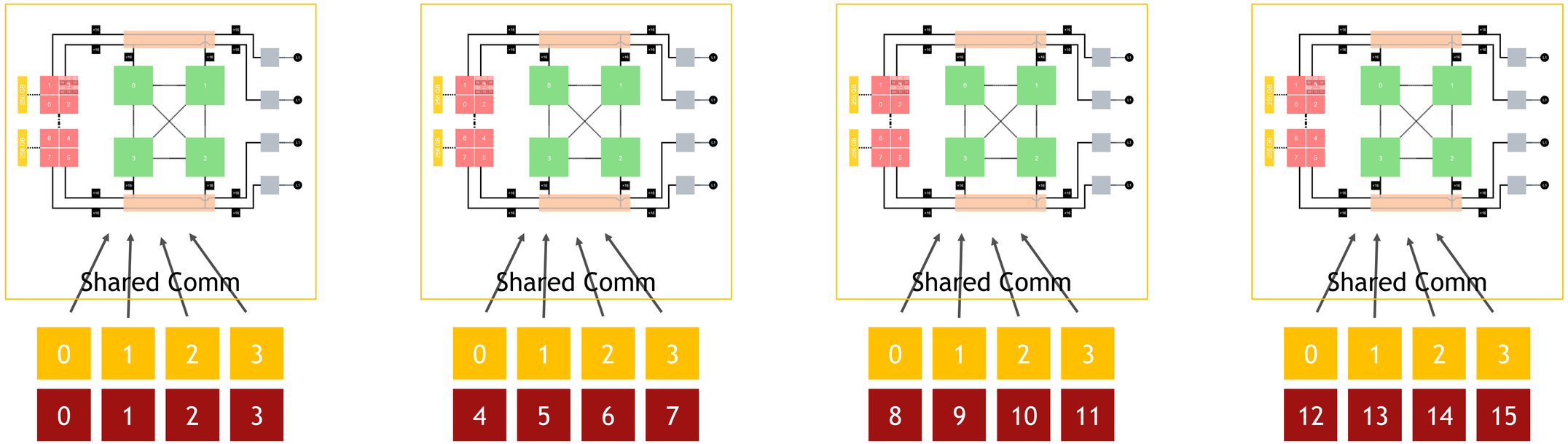


HANDLING MULTI GPU NODES

How to determine the local rank? – MPI-3

```
int local_rank = -1;

MPI_Comm local_comm;
MPI_Comm_split_type(MPI_COMM_WORLD, MPI_COMM_TYPE_SHARED, rank, MPI_INFO_NULL, &local_comm));

MPI_Comm_rank(local_comm, &local_rank);

MPI_Comm_free(&local_comm);

int num_devices = 0;

cudaGetDeviceCount(&num_devices);
cudaSetDevice(local_rank % num_devices); // modulo just for cases when more ranks than GPUs
```

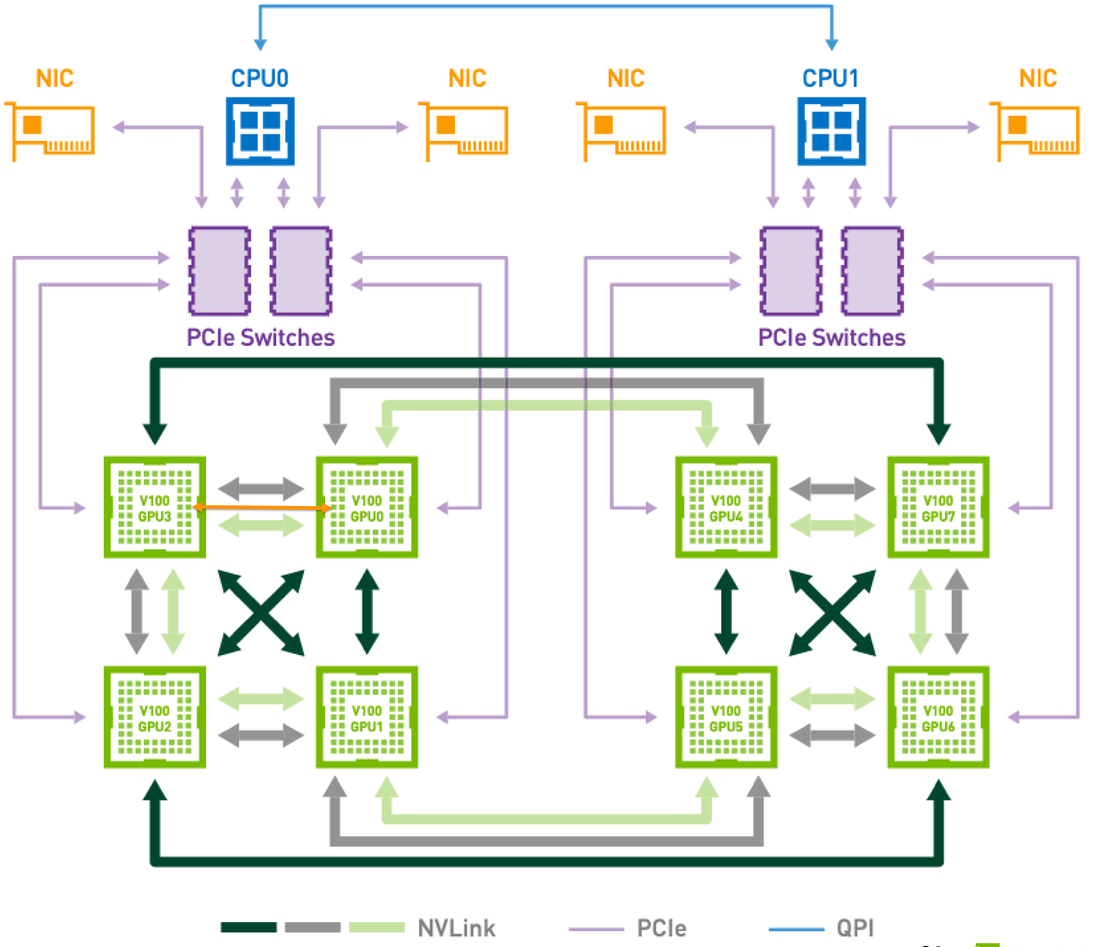
HANDLING MULTI GPU NODES



SOME TERMINOLOGY

P2P, CUDA aware, GPU Direct RDMA

P2P: direct exchange between GPUs



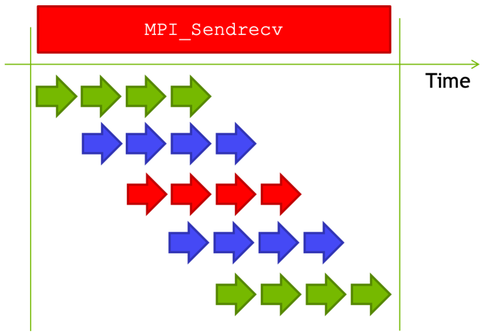
SOME TERMINOLOGY

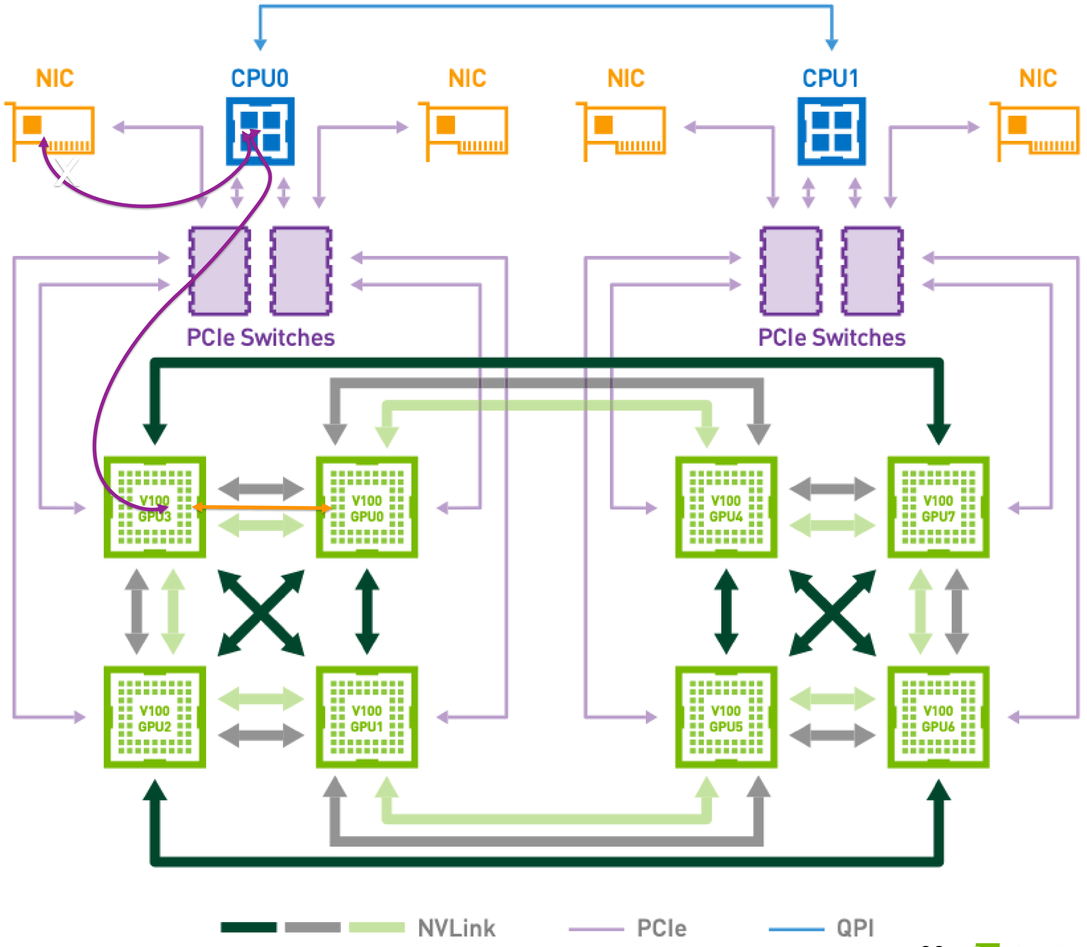
P2P, CUDA aware, GPU Direct RDMA

P2P: direct exchange between GPUs

CUDA aware MPI: MPI can take GPU pointer

can do batched, overlapping transfers (high bandwidth)





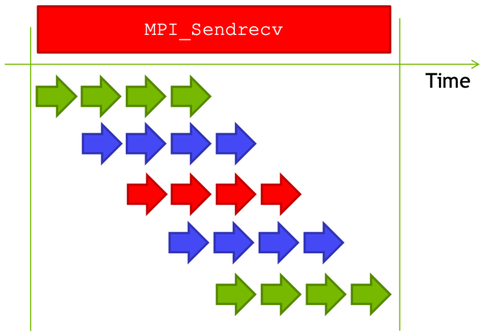
SOME TERMINOLOGY

P2P, CUDA aware, GPU Direct RDMA

P2P: direct exchange between GPUs

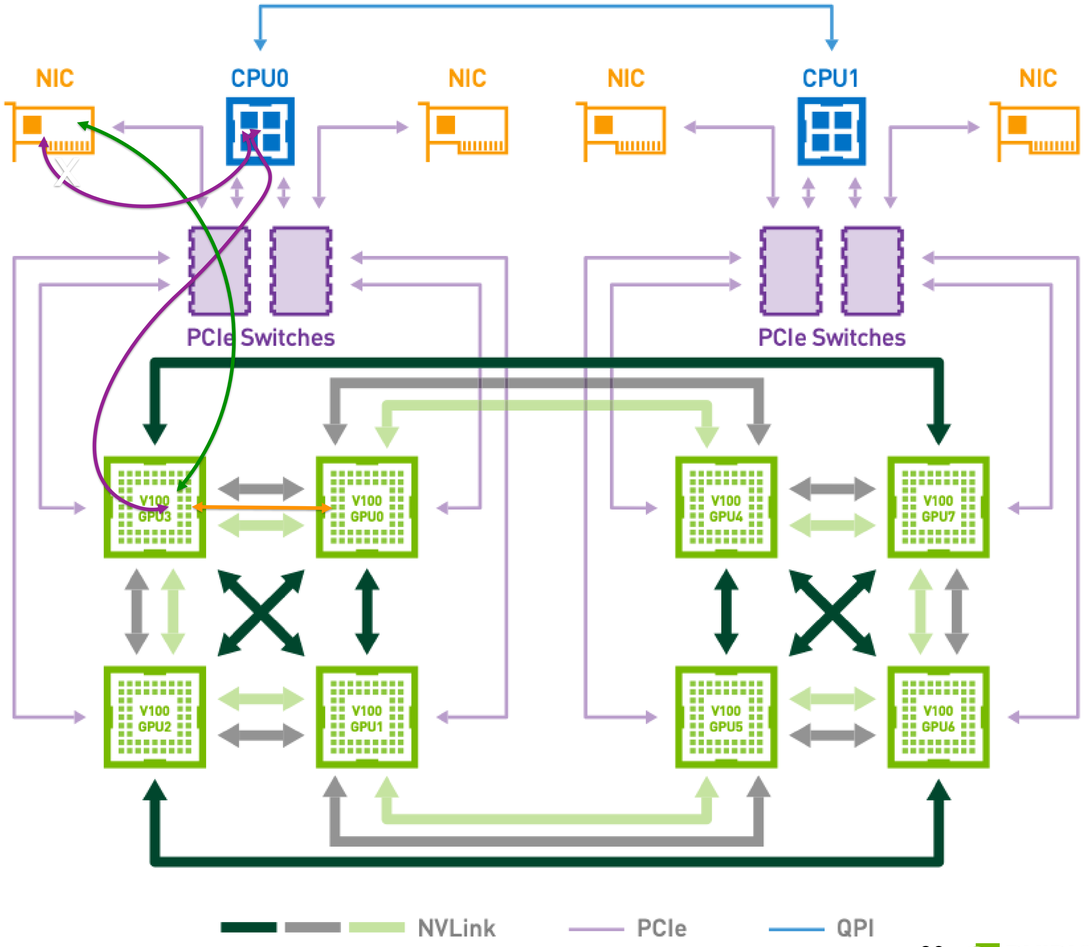
CUDA aware MPI: MPI can take GPU pointer

can do batched, overlapping transfers (high bandwidth)



GPU Direct RDMA: (implies CUDA aware MPI)

RDMA transfer from/to GPU memory here NIC <-> GPU

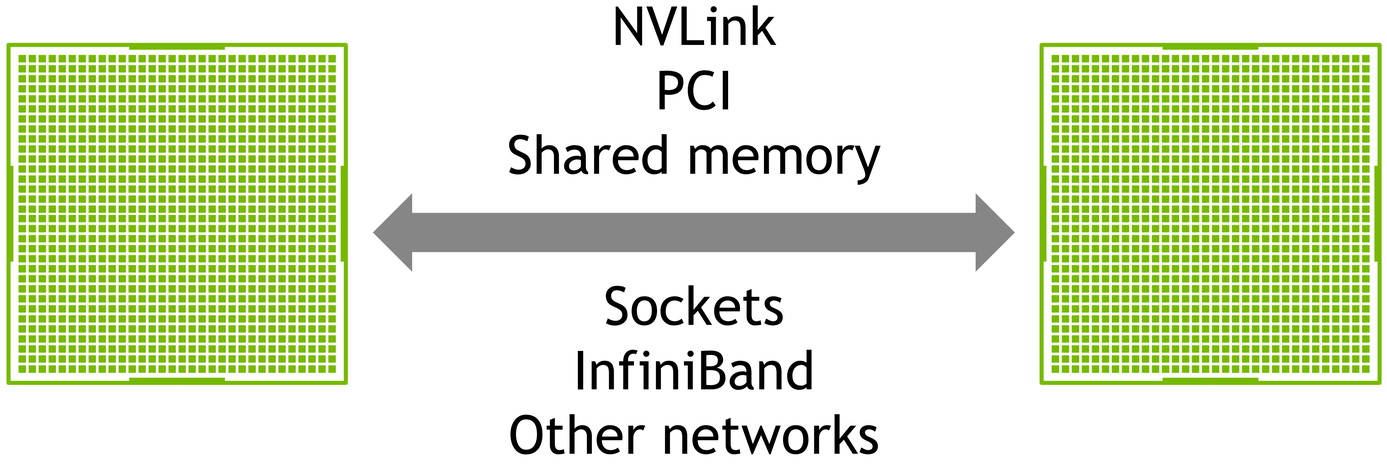


NCCL

Optimized inter-GPU communication

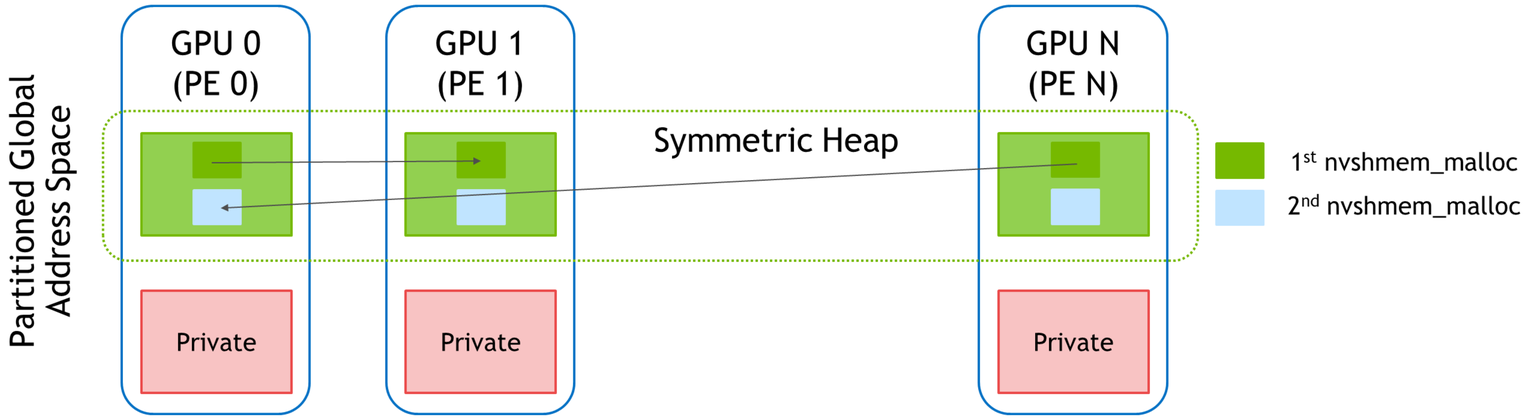
NCCL : NVIDIA Collective Communication Library

Communication library running on GPUs, for GPU buffers.



Binaries : https://developer.nvidia.com/nccl and in NGC containers
Source code : https://github.com/nvidia/nccl
Perf tests : https://github.com/nvidia/nccl-tests

NVSHMEM



Implementation of OpenSHMEM, a Partitioned Global Address Space (PGAS) library

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

Symmetric memory: `nvshmem_malloc(…);` Private memory: `cudaMalloc(…);`

CPU (blocking and stream-ordered) and CUDA Kernel interfaces

Read: `nvshmem_get(…);` Write: `nvshmem_put(…);` Atomic: `nvshmem_atomic_add(…);`

Flush writes: `nvshmem_quiet();` Order writes: `nvshmem_fence();`

Synchronize: `nvshmem_barrier();` Poll: `nvshmem_wait_until(…);`

Interoperable with MPI


NVIDIA