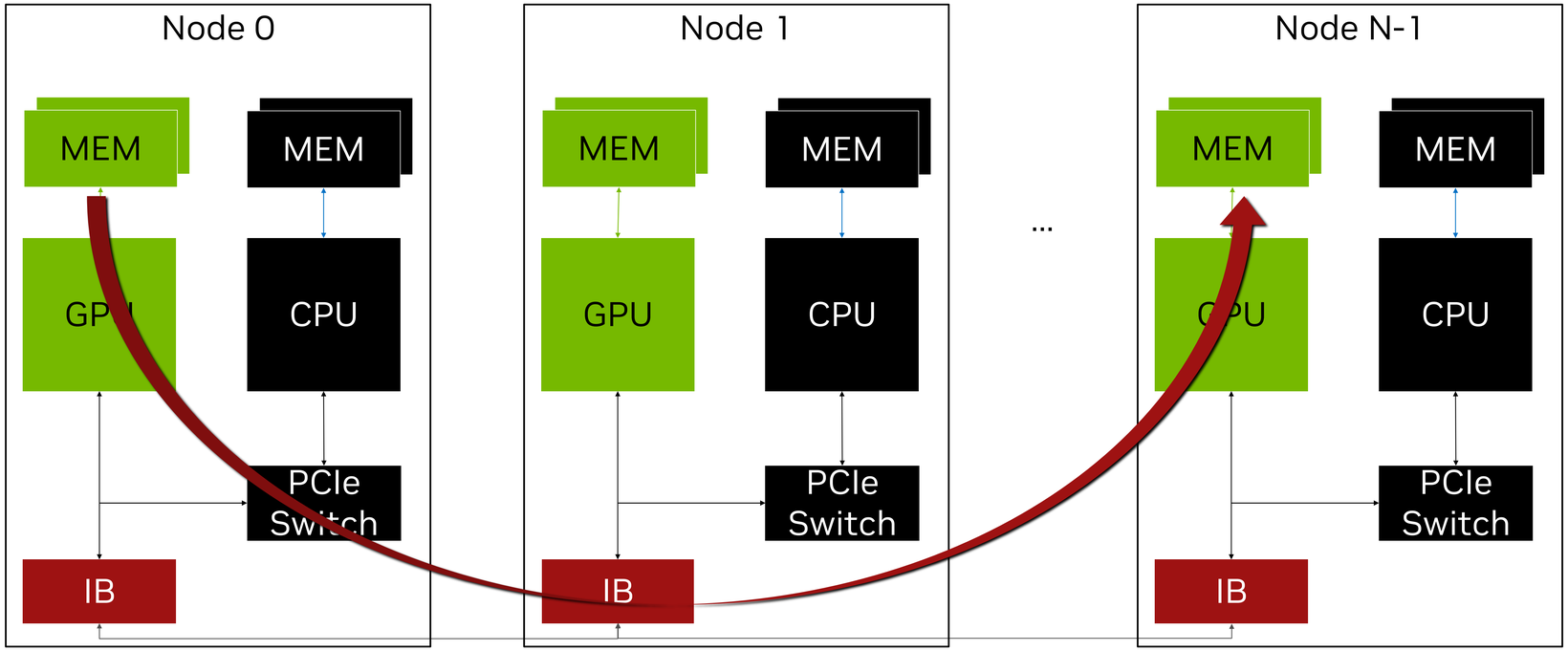
Multi GPU Programming with CUDA and MPI

Jiri Kraus, Principal Devtech Compute | April 10th 2024

MPI+CUDA




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

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

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, …

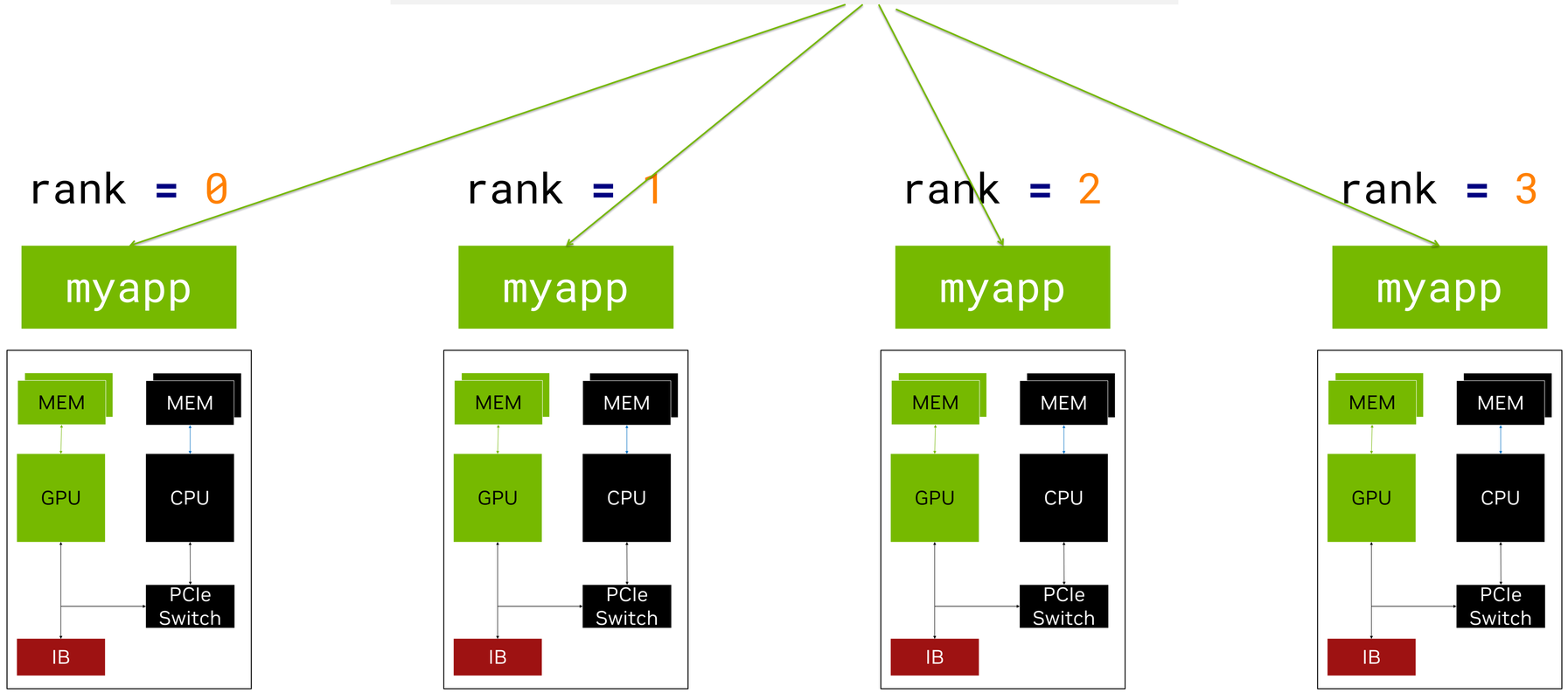
MPI - Skeleton

```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;
}
```

MPI

Compiling and Launching

```
$ mpicc –o myapp myapp.c
$ srun -n 4 ./myapp <args>
```



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) on left and right boundary

Periodic boundary conditions on top and bottom boundary

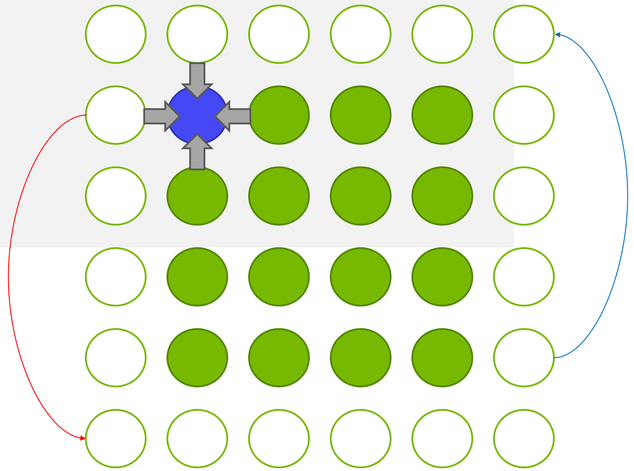
Example: Jacobi Solver

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

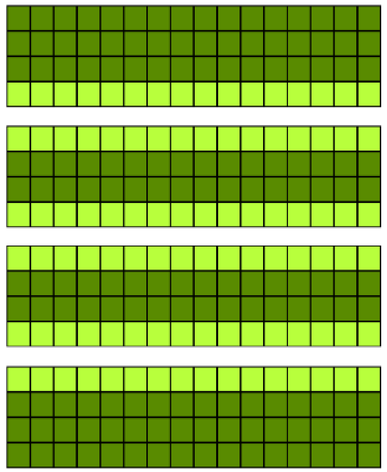
Swap a_new and a

Next iteration

Domain Decomposition

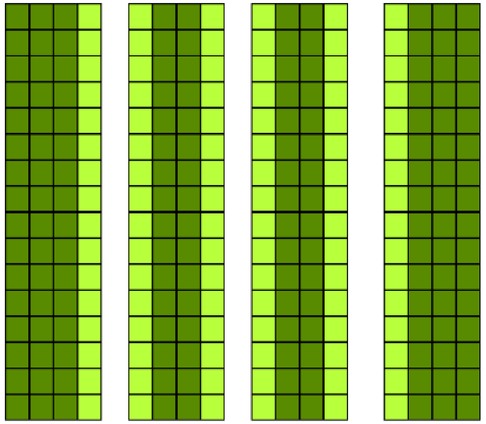
Different ways to split the work between processes:

- Minimize number of neighbors:
  - Communicate to less neighbors
  - Optimal for latency bound communication
- Minimize surface area/volume ratio:
  - Communicate less data
  - Optimal for bandwidth bound communication



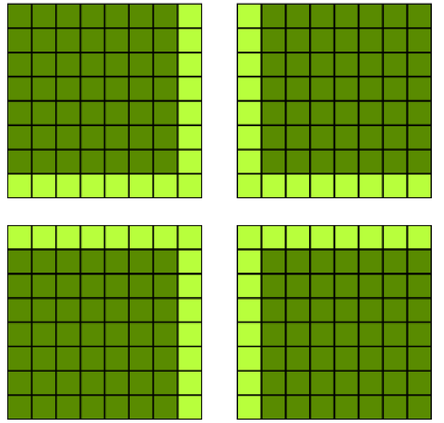
Horizontal Stripes

Contiguous if data is row-major



Vertical Stripes

Contiguous if data is column-major



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  ] );
```

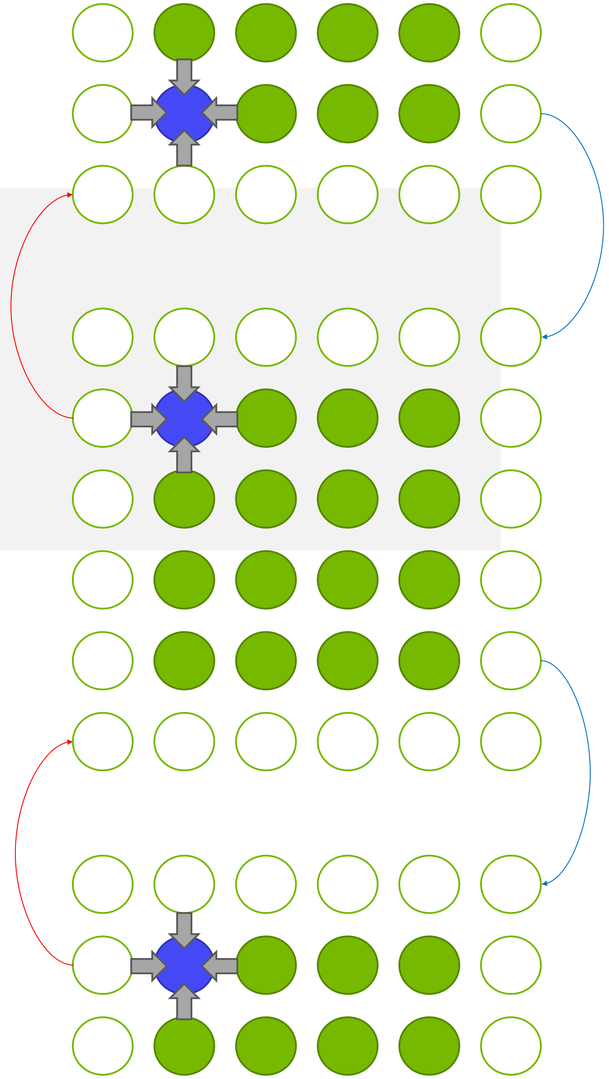
Apply periodic boundary conditions

Halo exchange

One-step with ring exchange

Swap a_new and a

Next iteration



Multi Process Multi GPU Programming

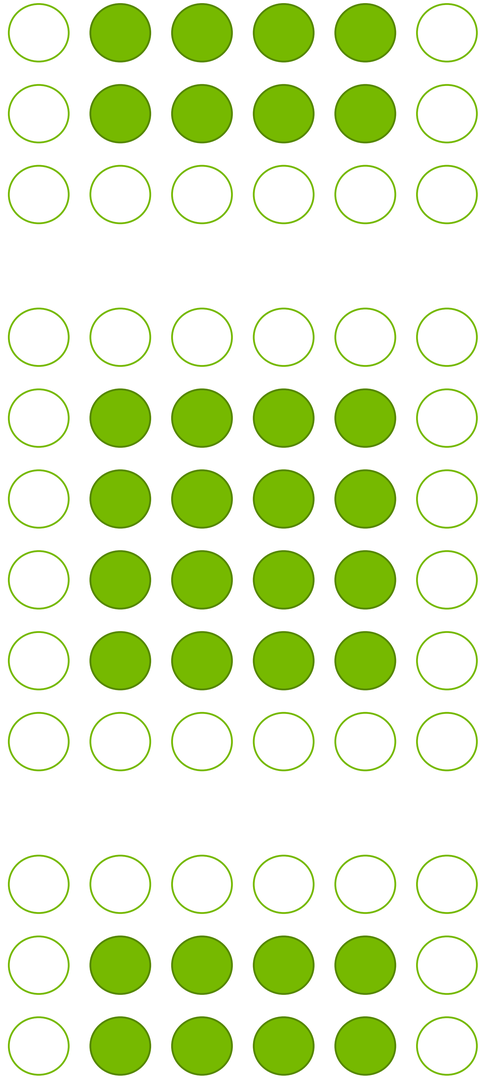
Using CUDA-aware MPI

```
while (l2_norm > tol && iter < iter_max) {
    cudaMemsetAsync(l2_norm_d, 0, sizeof(real), compute_stream);
    launch_jacobi_kernel(a_new, a, l2_norm_d, iy_start, iy_end, nx, compute_stream);
    cudaEventRecord(compute_done, compute_stream);
    cudaMemcpyAsync(l2_norm_h, l2_norm_d, sizeof(real), cudaMemcpyDeviceToHost, compute_stream);

    cudaEventSynchronize(compute_done);
    const int top = rank > 0 ? rank - 1 : (size - 1);
    const int bottom = (rank + 1) % size;
    // Top/Bottom Halo exchange -> next slide


    cudaStreamSynchronize(compute_stream);
    MPI_CALL(MPI_Allreduce(l2_norm_h, &l2_norm, 1, MPI_REAL_TYPE, MPI_SUM, MPI_COMM_WORLD));
    l2_norm = std::sqrt(l2_norm);

    std::swap(a_new, a); iter++;
}
```

Example Jacobi

Top/Bottom Halo

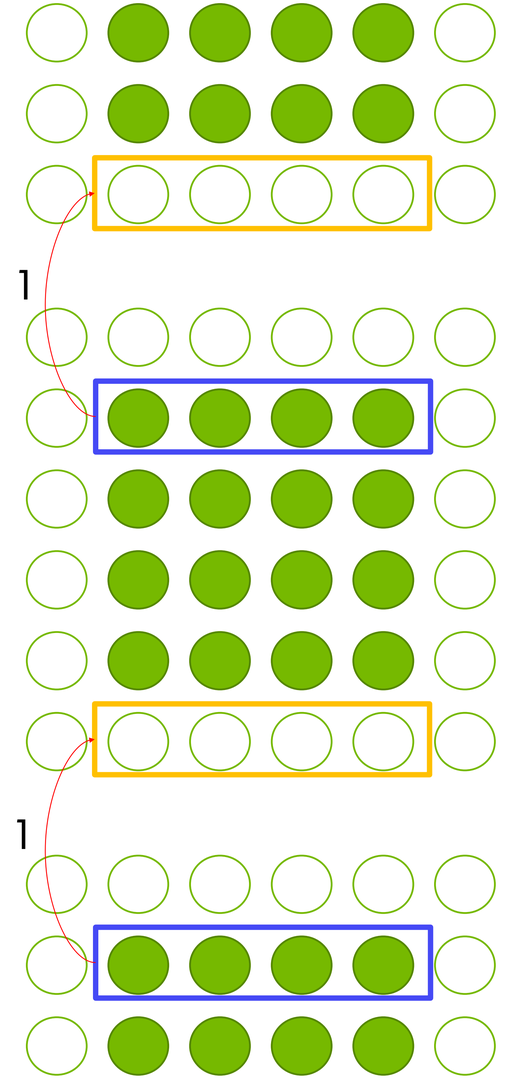
```
MPI_Sendrecv(a_new+iy_start*nx, nx, MPI_FLOAT, top , 0,
             a_new+(iy_end*nx), nx, MPI_FLOAT, bottom, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```



Example Jacobi

Top/Bottom Halo

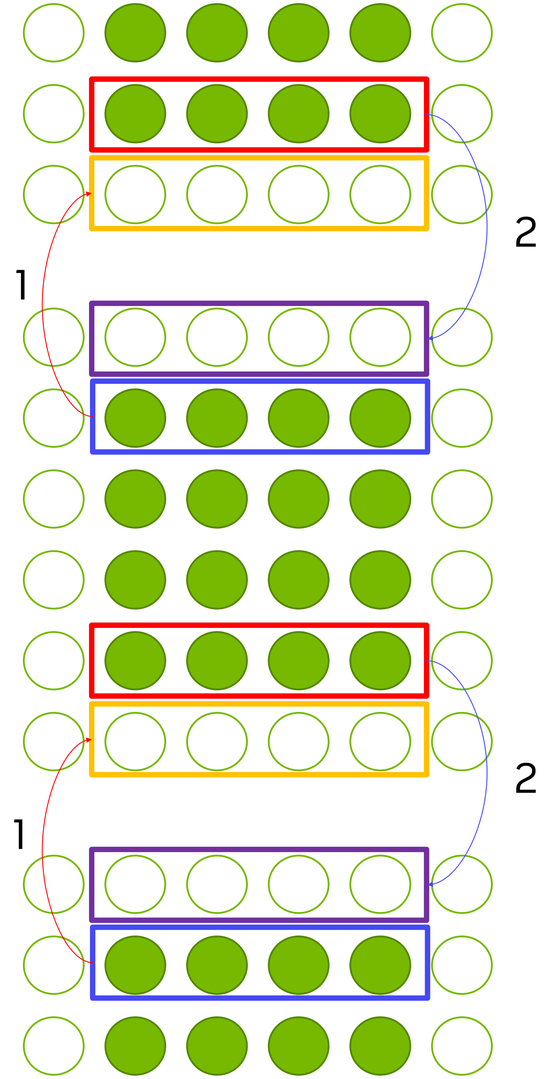
```
MPI_Sendrecv(a_new+iy_start*nx, nx, MPI_FLOAT, top , 0,
             a_new+(iy_end*nx), nx, MPI_FLOAT, bottom, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```



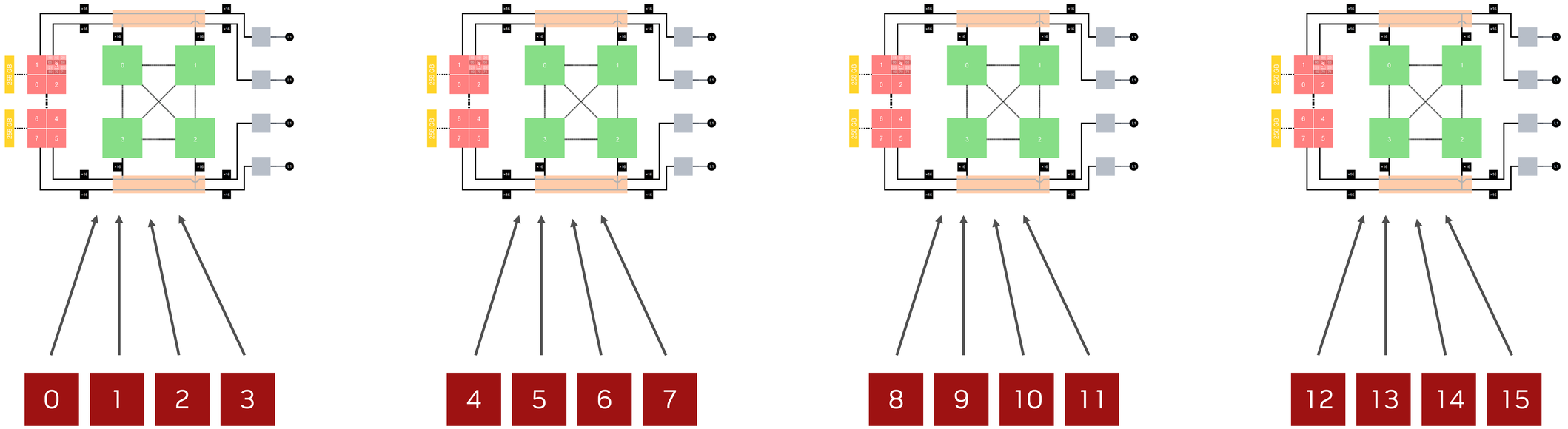
Example Jacobi

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);
```



Handling Multi GPU nodes

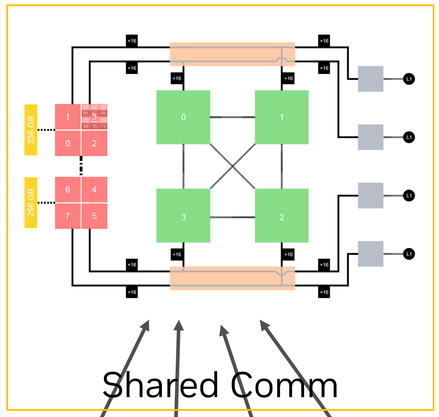


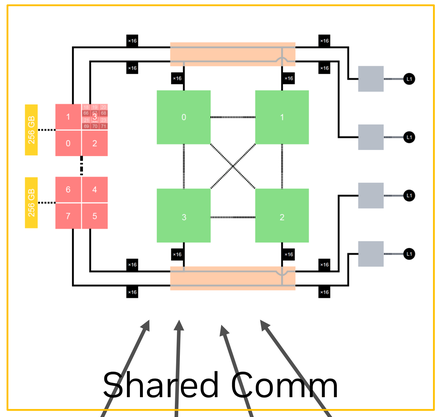
Handling Multi GPU nodes

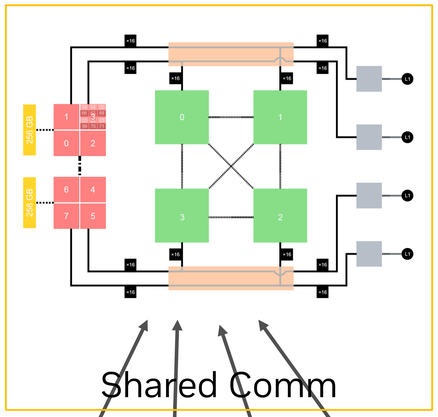
How to determine the local rank? – MPI-3

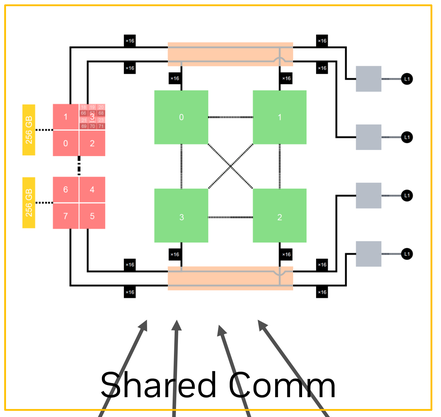
```
MPI_Comm local_comm;

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


MPI_CALL(MPI_Comm_rank(local_comm, &local_rank));


MPI_CALL(MPI_Comm_free(&local_comm));
```

Handling Multi GPU nodes









Multi Process Multi GPU Programming

Using CUDA-aware MPI

Handle GPU affinity on multi-GPU nodes:

```
int local_rank = -1;
MPI_Comm_rank(local_comm,&local_rank);
int num_devices = 0;
cudaGetDeviceCount(&num_devices);
cudaSetDevice(local_rank % num_devices);
```

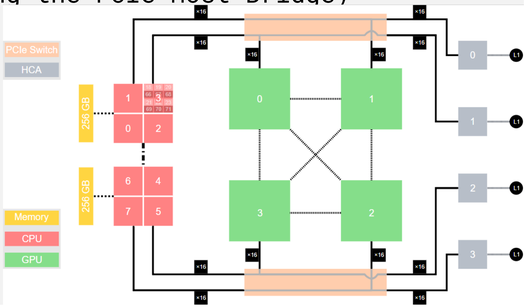
Needed if MPS is used or GPU affinity is handled by resource manager or launcher script.

Handling Multi GPU nodes

Example binding script

```
case ${SLURM_LOCALID} in
  0)
        export CUDA_VISIBLE_DEVICES=0
        export UCX_NET_DEVICES=mlx5_1:1
        CPU_BIND=18-23
        ;;
   1)
        export CUDA_VISIBLE_DEVICES=1
        export UCX_NET_DEVICES=mlx5_0:1
        CPU_BIND=6-11
        ;;
   2)
        export CUDA_VISIBLE_DEVICES=2
        export UCX_NET_DEVICES=mlx5_3:1
        CPU_BIND=42-47
        ;;
   3)
        export CUDA_VISIBLE_DEVICES=3
        export UCX_NET_DEVICES=mlx5_2:1
        CPU_BIND=30-35
        ;;
esac
numactl --physcpubind=${CPU_BIND} $*
```

```
[kraus1@jwb0007]$ nvidia-smi topo -m
        GPU0    GPU1    GPU2    GPU3    mlx5_0  mlx5_1  mlx5_2  mlx5_3  CPU Affinity    NUMA Affinity
GPU0     X      NV4     NV4     NV4     SYS     PIX     SYS     SYS     18-23,66-71     3
GPU1    NV4      X      NV4     NV4     PIX     SYS     SYS     SYS     6-11,54-59      1
GPU2    NV4     NV4      X      NV4     SYS     SYS     SYS     PIX     42-47,90-95     7
GPU3    NV4     NV4     NV4      X      SYS     SYS     PIX     SYS     30-35,78-83     5
mlx5_0  SYS     PIX     SYS     SYS      X      SYS     SYS     SYS
mlx5_1  PIX     SYS     SYS     SYS     SYS      X      SYS     SYS
mlx5_2  SYS     SYS     SYS     PIX     SYS     SYS      X      SYS
mlx5_3  SYS     SYS     PIX     SYS     SYS     SYS     SYS      X

Legend:

  X    = Self
  SYS  = Connection traversing PCIe as well as the SMP interconnect between NUMA nodes (e.g., QPI/UPI)
  NODE = Connection traversing PCIe as well as the interconnect between PCIe Host Bridges within a NUMA
node
  PHB  = Connection traversing PCIe as well as a PCIe Host Bridge (typically the CPU)
  PXB  = Connection traversing multiple PCIe bridges (without traversing the PCIe Host Bridge)
  PIX  = Connection traversing at most a single PCIe bridge
  NV#  = Connection traversing a bonded set of # NVLinks
```



https://apps.fz-juelich.de/jsc/hps/juwels/booster-overview.html

Approaches for multi-process tools

- Tools usually run on a single process – adapt for highly distributed applications?
  - Bugs in parallel programs are often serial bugs in disguise
- Common MPI paradigm: Workload distributed; bug classes/performance similar for all processes
  - Not: Load imbalance, parallel race conditions; require parallel tools
- Ergo: Run tool N times in parallel, have N output files, only look at 1 (or 2, …)
- `%q{ENV_VAR}` supported by all the NVIDIA tools discussed here, embed environment variable in file name
  - `ENV_VAR` should be one set by the process launcher, unique ID
  - Evaluated only once tool starts running (on compute node) – not when launching job
- Other tools: Use a launcher script, for late evaluation

OpenMPI:
`OMPI_COMM_WORLD_RANK`
`OMPI_COMM_WORLD_LOCAL_RANK`

MVAPICH2:
`MV2_COMM_WORLD_RANK`
`MV2_COMM_WORLD_LOCAL_RANK`

Slurm:
`SLURM_PROCID`
`SLURM_LOCALID`

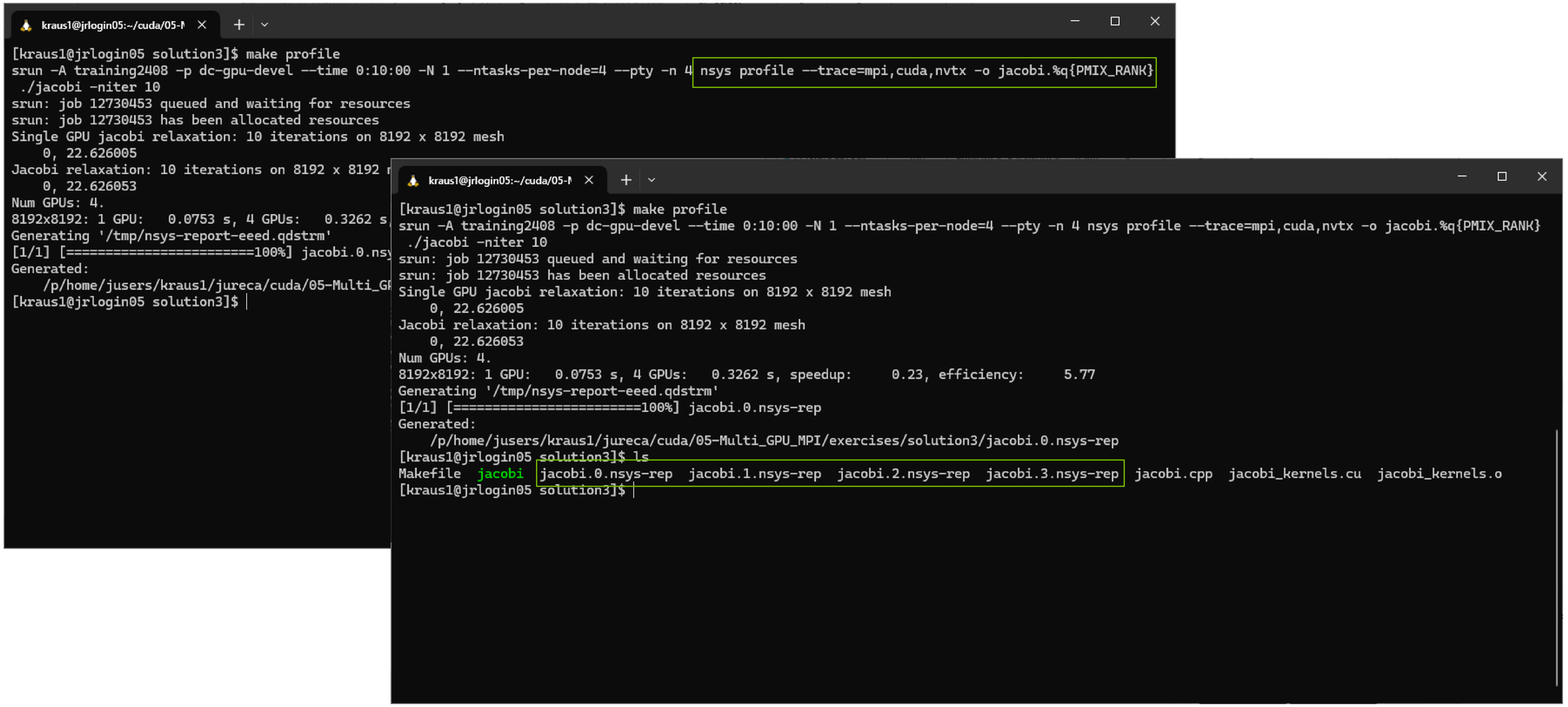
PMIX:
`PMIX_RANK`

https://www.open-mpi.org/faq/?category=running#mpi-environmental-variables
http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.2-userguide.html#x1-32100013
https://slurm.schedmd.com/srun.html#SECTION_OUTPUT-ENVIRONMENT-VARIABLES

Credit Markus Hrywniak

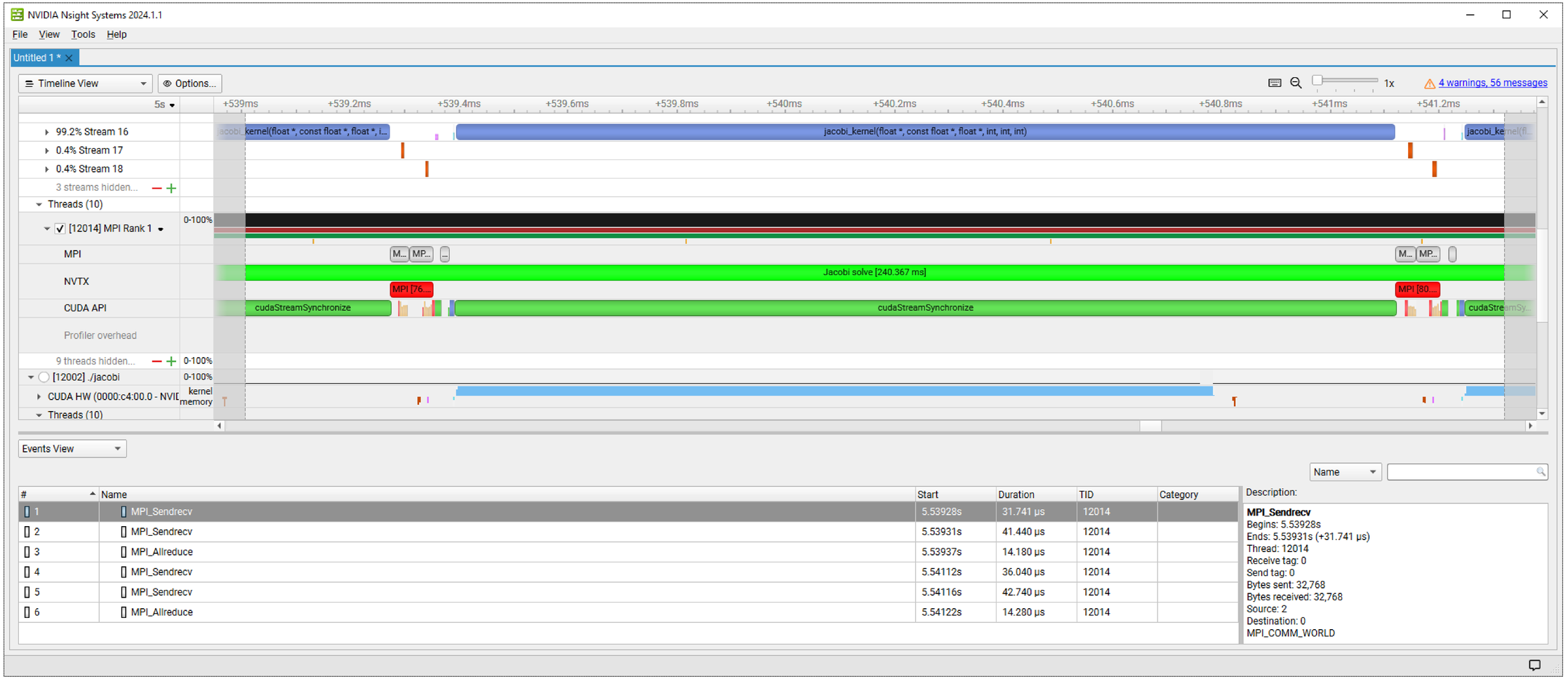
Profiling MPI+CUDA Applications

Using Nsight Systems



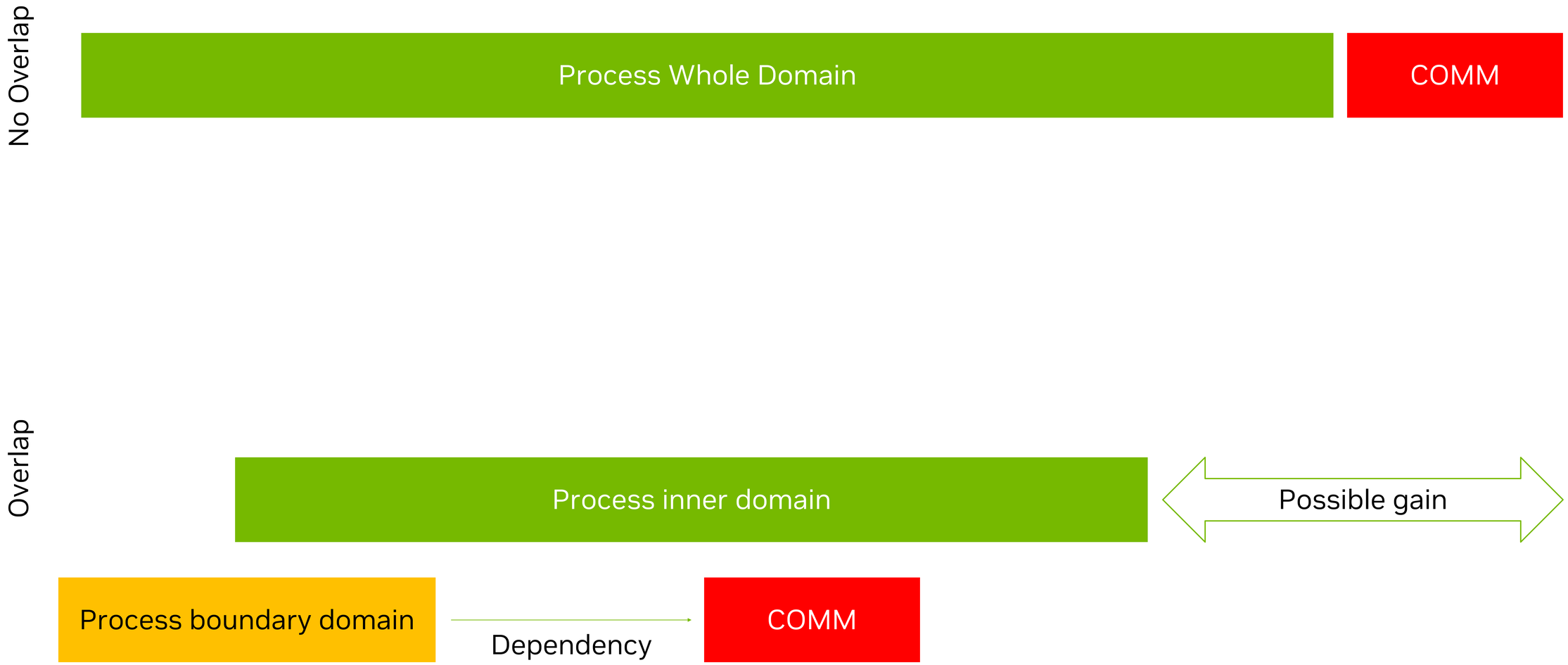
Profiling MPI+CUDA Applications

Using Nsight Systems



#	Name	Start	Duration	TID	Category
1	MPI_Sendrecv	5.53928s	31.741 µs	12014	
2	MPI_Sendrecv	5.53931s	41.440 µs	12014	
3	MPI_Allreduce	5.53937s	14.180 µs	12014	
4	MPI_Sendrecv	5.54112s	36.040 µs	12014	
5	MPI_Sendrecv	5.54116s	42.740 µs	12014	
6	MPI_Allreduce	5.54122s	14.280 µs	12014	

Overlapping Communication and Computation



MPI

Overlapping Communication and Computation

```
launch_jacobi_kernel( a_new, a, l2_norm_d, iy_start, (iy_start+1), nx, halo_stream );
launch_jacobi_kernel( a_new, a, l2_norm_d, (iy_end-1), iy_end, nx, halo_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( halo_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 );
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 );
```

Hands-on Menu

4 Tasks to choose rom

- Task 0: Using MPI
- Task 1: Handle GPU Affinity
- Task 2: Apply Domain Decomposition
- Task 3: Overlap MPI and Compute

Task 0: Using MPI

- Determine rank (`MPI_Comm_rank`) and size (`MPI_Comm_size`)
- Add `MPI_Barrier` to ensure correct timing

Make Targets:
`run`: run `jacobi` with `$NP` procs.
`jacobi`: build `jacobi` bin (default)
`sanitize`: run with `compute-sanitizer`
`profile`: profile with Nsight Systems
Solution is in `solution0`

https://www.open-mpi.org/doc/current/

Task 1: Handling GPU affinity

- Run with `CUDA_VISIBLE_DEVICES=0,1,2,3`
- Handle GPU affinity with `MPI_COMM_TYPE_SHARED`
- Run and report the performance

Make Targets:
`run`: run `jacobi` with `$NP` procs.
`jacobi`: build `jacobi` bin (default)
`sanitize`: run with `compute-sanitizer`
`profile`: profile with Nsight Systems
Solution is in `solution1`

https://www.open-mpi.org/doc/current/

Task 2: Apply Domain Decomposition

- Calculate first (`iy_start`) and last (`iy_end`) row to be processed by each rank
- Use `MPI_Sendrecv` to handle halo updates and periodic boundary conditions
- Use `MPI_Allreduce` to calculate global L2 norm

Make Targets:
`run`: run `jacobi` with $NP procs.
`jacobi`: build `jacobi` bin (default)
`sanitize`: run with `compute-sanitizer`
`profile`: profile with Nsight Systems
Solution is in `solution2`

https://www.open-mpi.org/doc/current/

Task 3: Overlap MPI and Compute

- Use `cudaStreamCreate` to create halo processing stream
- Split jacobi step in top boundary, bottom boundary and bulk part
- Launch top and bottom boundary part in halo processing stream

Make Targets:
`run`: run `jacobi` with `$NP` procs.
`jacobi`: build `jacobi` bin (default)
`sanitize`: run with `compute-sanitizer`
`profile`: profile with Nsight Systems
Solution is in `solution3`

https://www.open-mpi.org/doc/current/

Task 0: Using MPI

Solution

```
int main(int argc, char * argv[]) {
    int rank = 0;
    int size = 1;
    MPI_CALL( MPI_Init(&argc,&argv) );
    MPI_CALL( MPI_Comm_rank(MPI_COMM_WORLD,&rank) );
    MPI_CALL( MPI_Comm_size(MPI_COMM_WORLD,&size) );
    //…
    MPI_CALL( MPI_Barrier(MPI_COMM_WORLD) );
    double start = MPI_Wtime();
    while ( l2_norm > tol && iter < iter_max )
    //…
    MPI_CALL( MPI_Finalize() );
    return result_correct == 1 ? 0 : 1; }
```

Task 1: Handling GPU affinity

Solution

```
int dev_id = -1;
MPI_Comm local_comm;
MPI_Info info;
MPI_CALL( MPI_Info_create(&info) );
MPI_CALL( MPI_Comm_split_type(MPI_COMM_WORLD, MPI_COMM_TYPE_SHARED,
                              rank, info, &local_comm) );
MPI_CALL( MPI_Comm_rank(local_comm,&dev_id) );
MPI_CALL( MPI_Comm_free(&local_comm) );
MPI_CALL( MPI_Info_free(&info) );
int num_devs = 0;
CUDA_RT_CALL( cudaGetDeviceCount( &num_devs ) );
dev_id = dev_id % num_devs;
CUDA_RT_CALL( cudaSetDevice( dev_id ) );
```

Task 2: Apply Domain Decomposition

Solution I/III

```
// Ensure correctness if ny%size != 0
int chunk_size = std::ceil( (1.0*ny)/size );
int iy_start = rank*chunk_size;
int iy_end = iy_start+chunk_size;
// Do not process boundaries
iy_start = std::max( iy_start, 1 );
iy_end = std::min( iy_end, ny - 1 );
```

Task 2: Apply Domain Decomposition

Solution II/III

```
//Apply periodic boundary conditions
CUDA_RT_CALL( cudaStreamSynchronize( compute_stream ) );
PUSH_RANGE("MPI",5)
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 ));
MPI_CALL( MPI_Sendrecv( a_new+(iy_end-1)*nx, nx, MPI_REAL_TYPE, bottom, 0,
                        a_new+(iy_start-1)*nx, nx, MPI_REAL_TYPE, top, 0,
                        MPI_COMM_WORLD, MPI_STATUS_IGNORE ));
POP_RANGE
```

Task 2: Apply Domain Decomposition

Solution III/III

```
CUDA_RT_CALL( cudaStreamSynchronize( compute_stream ) );
MPI_CALL( MPI_Allreduce( l2_norm_m, &l2_norm, 1, MPI_REAL_TYPE,
                         MPI_SUM, MPI_COMM_WORLD ) );
l2_norm = std::sqrt( l2_norm );
```

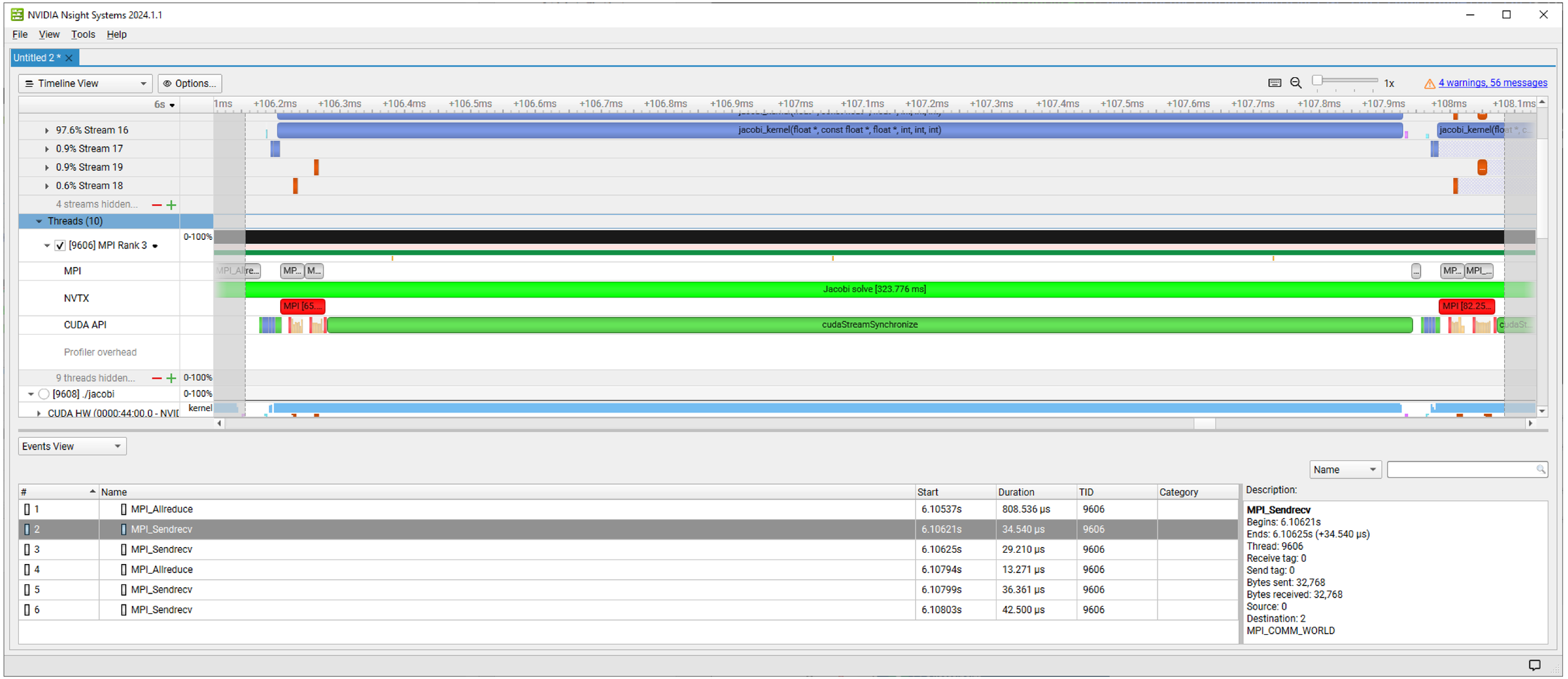
Task 3: Overlap MPI and Compute

Solution

```
launch_jacobi_kernel( a_new, a, l2_norm_d,
                      iy_start, (iy_start+1), nx, halo_stream );
launch_jacobi_kernel( a_new, a, l2_norm_d,
                      (iy_end-1), iy_end, nx, halo_stream );
launch_jacobi_kernel( a_new, a, l2_norm_d, (iy_start+1),
                      (iy_end-1), nx, compute_stream );
int top = rank > 0 ? rank - 1 : (size-1); int bottom = (rank+1)%size;
//Apply periodic boundary conditions
CUDA_RT_CALL( cudaStreamSynchronize( halo_stream ) );
PUSH_RANGE("MPI",5)
MPI_CALL( MPI_Sendrecv( a_new+iy_start*nx, …
```

Task 3: Overlap MPI and Compute

Solution



#	Name	Start	Duration	TID	Category
1	MPI_Allreduce	6.10537s	808.536 µs	9606	
2	MPI_Sendrecv	6.10621s	34.540 µs	9606	
3	MPI_Sendrecv	6.10625s	29.210 µs	9606	
4	MPI_Allreduce	6.10794s	13.271 µs	9606	
5	MPI_Sendrecv	6.10799s	36.361 µs	9606	
6	MPI_Sendrecv	6.10803s	42.500 µs	9606