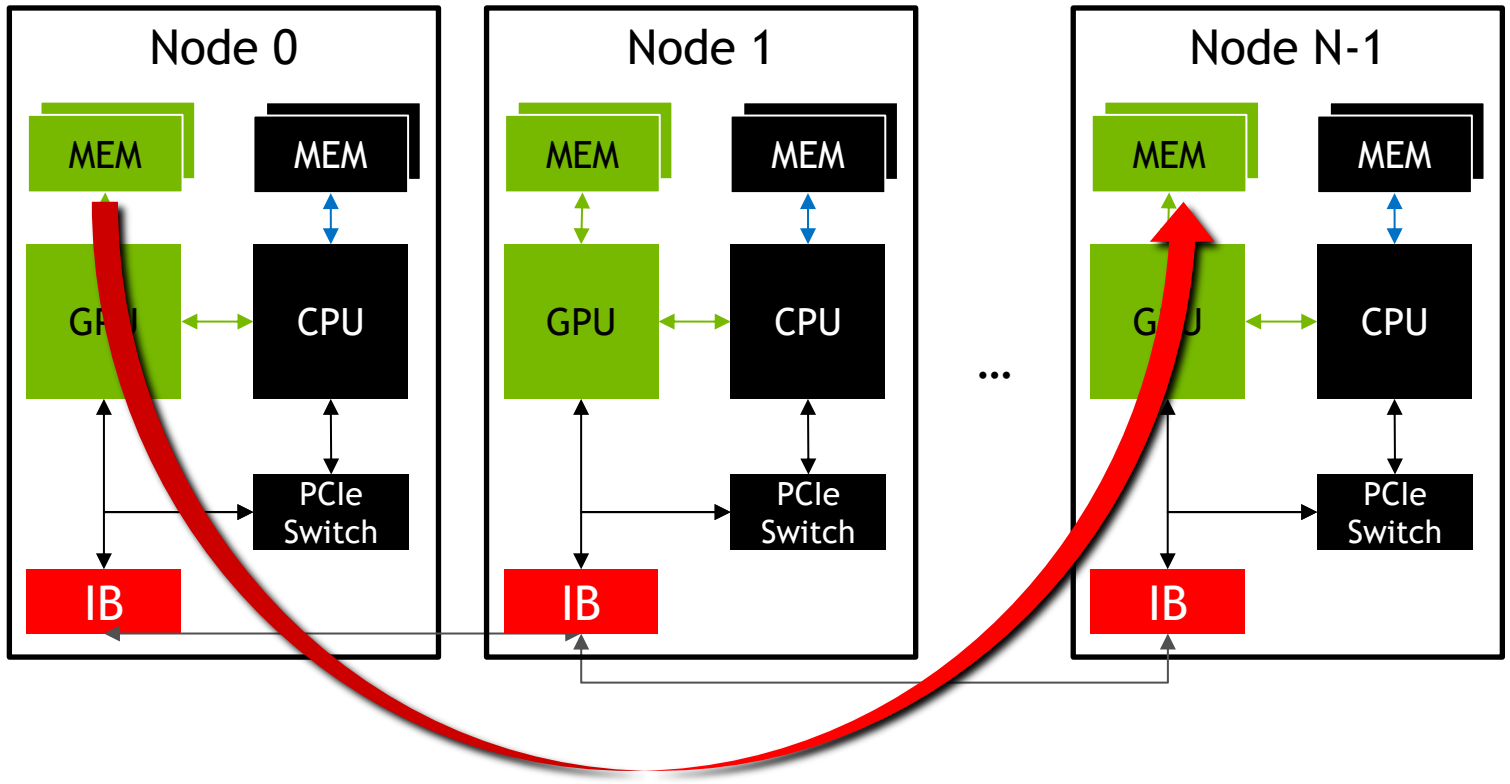
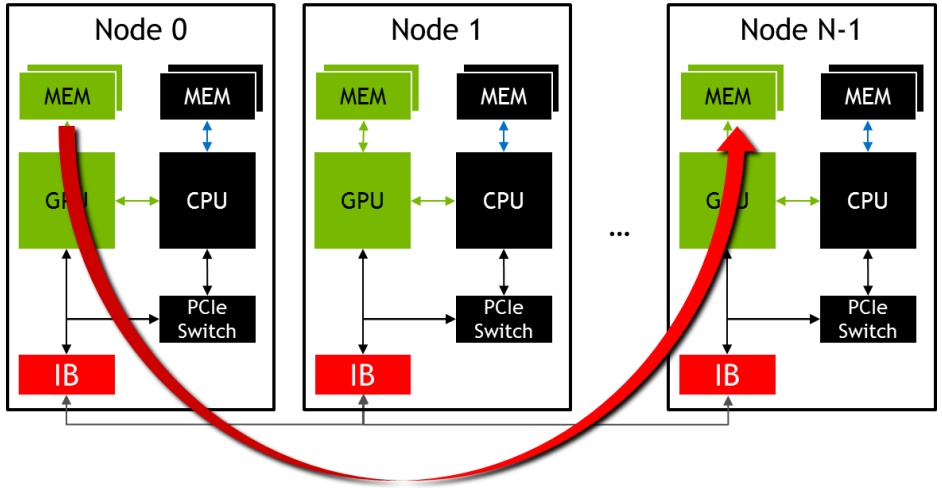
MULTI GPU PROGRAMMING WITH MPI AND OPENACC

31.10.2023 I JIRI KRAUS (NVIDIA)

MPI+OPENACC



MPI+OPENACC




```
//MPI rank 0
#pragma acc host_data use_device( sbuf )
MPI_Send(sbuf, size, MPI_DOUBLE, n-1, tag, MPI_COMM_WORLD);

//MPI rank n-1
#pragma acc host_data use_device( rbuf )
MPI_Recv(rbuf, size, MPI_DOUBLE, 0, tag, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

WHAT YOU WILL LEARN

- What MPI is
- How to use MPI for inter GPU communication with OpenACC
- How to use Nsight Systems for MPI+OpenACC applications
- How to hide MPI communication times

MESSAGE PASSING INTERFACE - MPI

- Standard to exchange data between processes via messages
  - Defines API to exchanges messages
    - Pt. 2 Pt.: e.g. MPI_Send, MPI_Recv
    - Collectives, e.g. MPI_Allreduce
- Multiple implementations (open source and commercial)
  - Binding for C/C++, Fortran, Python, …
  - E.g. MPICH, OpenMPI, Parastation MPI, MVAPICH, IBM Platform MPI, Cray MPT, …

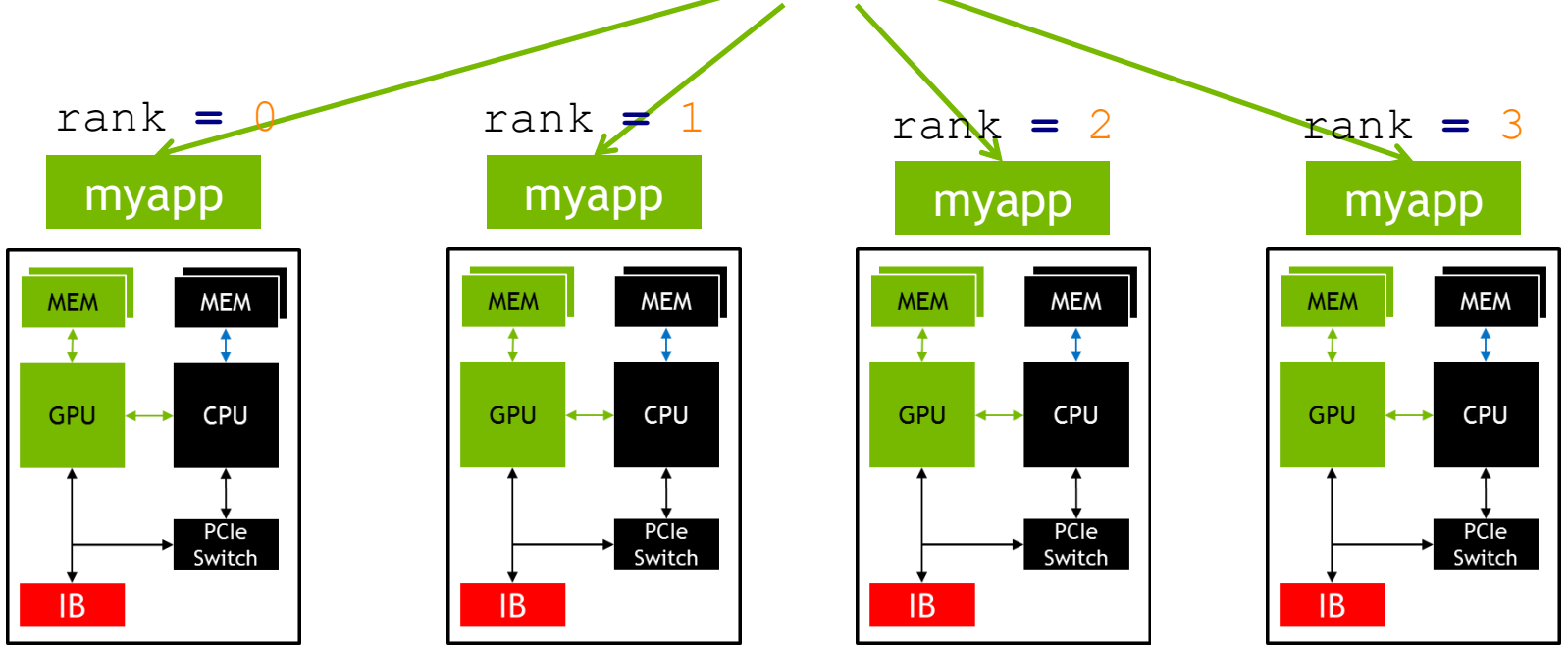
MPI – A MINIMAL PROGRAM

```c
#include <mpi.h>
int main(int argc, char *argv[]) {
      int rank,size;
      /* Initialize the MPI library */
      MPI_Init(&argc,&argv);
      /* Determine the calling 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;
}
```

Remark: Almost all MPI routines return an error value which should be checked. The examples and tasks leave that out for brevity.

MPI – COMPILING AND LAUNCHING

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

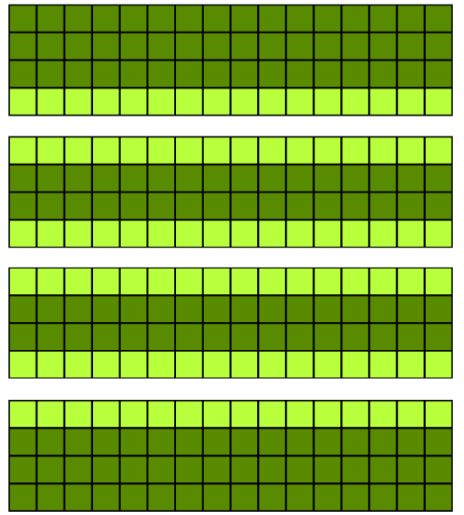


EXAMPLE: JACOBI SOLVER

Solves the 2D-Poission Equation on a rectangle

Periodic boundary conditions

Domain decomposition with stripes



EXAMPLE: JACOBI SOLVER – SINGLE GPU

While not converged

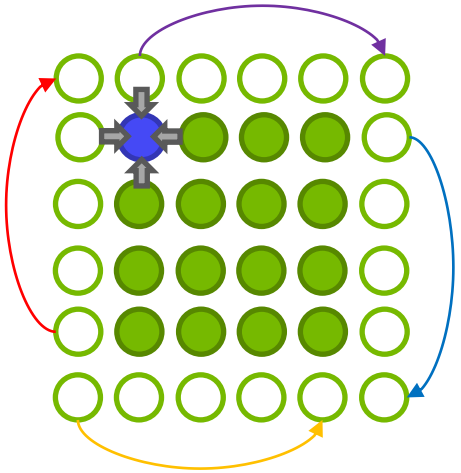
Do Jacobi step:

```
for (int iy = 1; iy < ny-1; ++iy)

for (int ix = 1; ix < nx-1; ++ix)

  Anew[iy*nx+ix]=-0.25f*(rhs[iy*nx+ix]

                  -(A[iy*nx+(ix-1)]+A[iy*nx+(ix+1)]

                   +A[(iy-1)*nx+ix]+A[(iy+1)*nx+ix]));
```

Copy `Anew` to `A`

Apply periodic boundary conditions

Next iteration

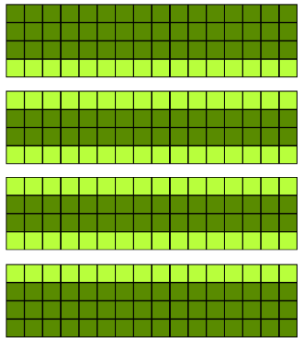


DOMAIN DECOMPOSITION

Different ways to split the work between processes:

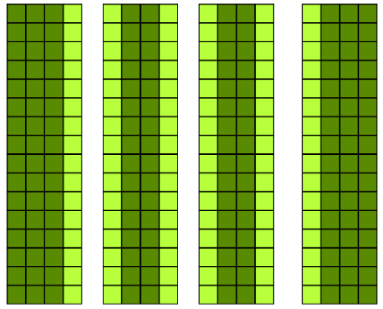
Minimizes number of neighbors:

- Communicate to less neighbors
- Optimal for latency bound communication



Horizontal Stripes

Contiguous if data is row-major

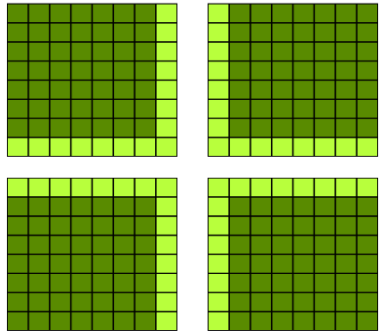


Vertical Stripes

Contiguous if data is column-major

Minimizes surface area/volume ratio:

- Communicate less data
- Optimal for bandwidth bound communication



Tiles

EXAMPLE: JACOBI SOLVER – MULTI GPU

While not converged

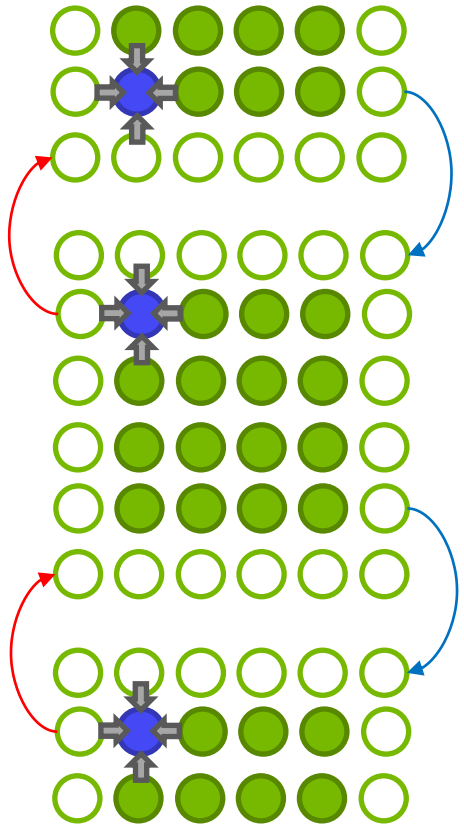
Do Jacobi step:

```
for (int iy = iy_start; iy < iy_end; ++iy)
for (int ix = 1; ix < NX-1; ++ix)
  Anew[iy*nx+ix]=-0.25f*(rhs[iy*nx+ix]
                  -(A[iy*nx+(ix-1)]+A[iy*nx+(ix+1)]
                   +A[(iy-1)*nx+ix]+A[(iy+1)*nx+ix]));
```

Copy `Anew` to `A`

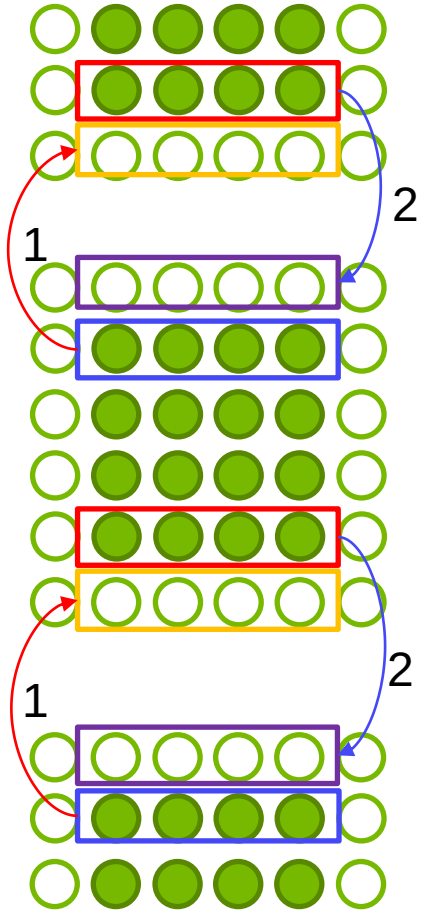
Apply periodic boundary conditions and exchange halo with 2 neighbors

Next iteration

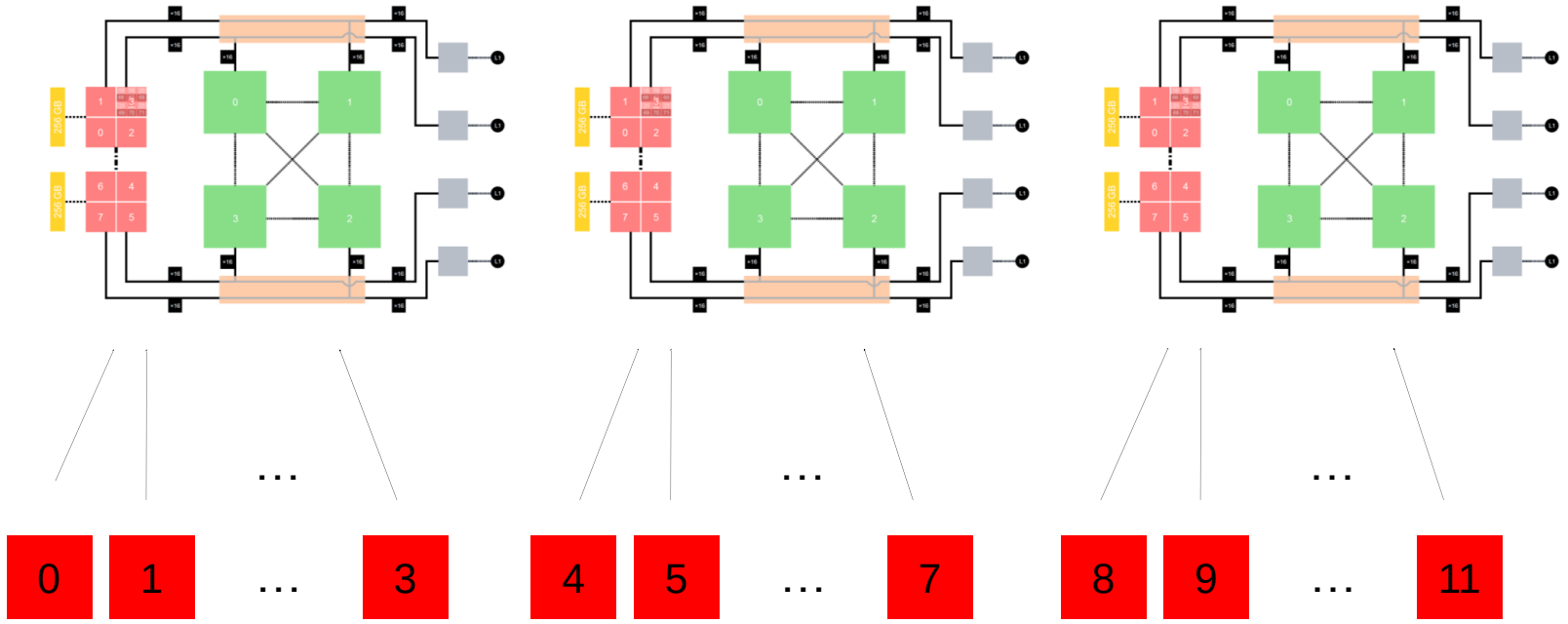


EXAMPLE: JACOBI – TOP/BOTTOM HALO

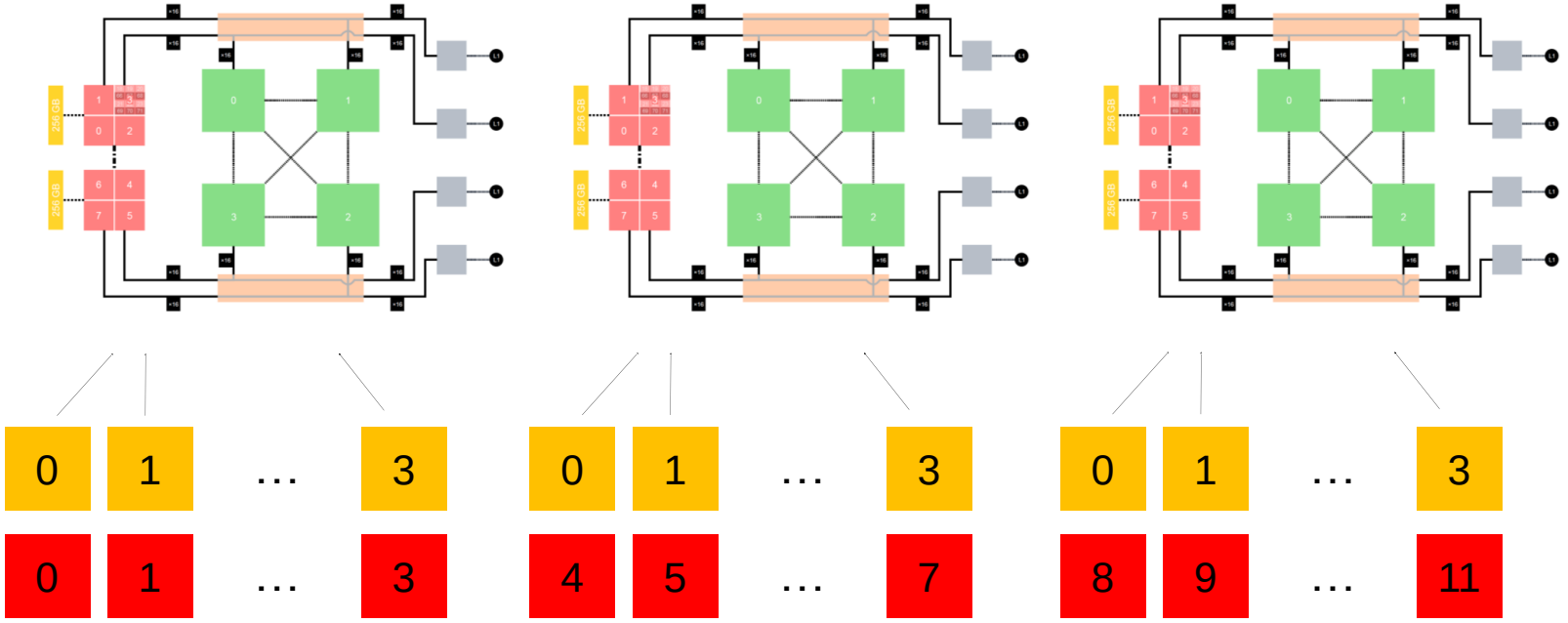
```
#pragma acc host_data use_device ( A ) {
MPI_Sendrecv(A+iy_start*nx+1, nx-2, MPI_DOUBLE, top, 0,
             A+iy_end*nx+1, nx-2, MPI_DOUBLE, bottom, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);

MPI_Sendrecv(A+(iy_end-1)*nx+1, nx-2, MPI_DOUBLE, bottom, 1,
             A+(iy_start-1)*nx+1, nx-2, MPI_DOUBLE, top, 1,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}
```



HANDLING GPU AFFINITY



HANDLING GPU AFFINITY



HANDLING GPU AFFINITY

```
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);
}
#ifdef _OPENACC
int num_devs = acc_get_num_devices(acc_get_device_type());
#else
int num_devs = 1;
#endif
#pragma acc set device_num( local_rank%num_devs )
```

Needed if resource manager handles GPU affinity.

Handled gracefully on JUWELS-Booster by NVHPC if omitted:
"*If the value of* `devicenum` *is greater than or equal to the value returned by* `acc_get_num_devices` *for that device type, the behavior is implementation-defined.*"

PROFILING MPI+OPENACC APPLICATIONS

Embed MPI rank in output filename, process name, and context name

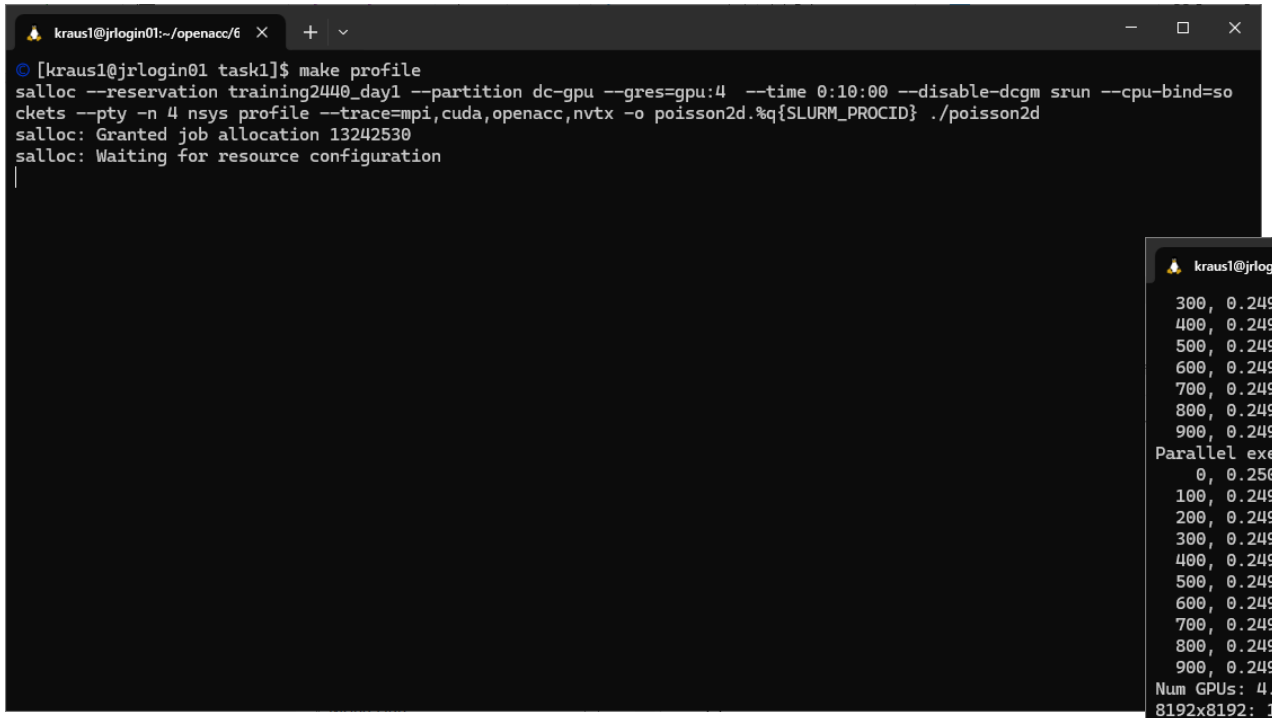
```
srun nsys profile --trace=mpi,cuda,openacc,nvtx          \
                  --output profile.%q{SLURM_PROCID}      \
                ./application
```

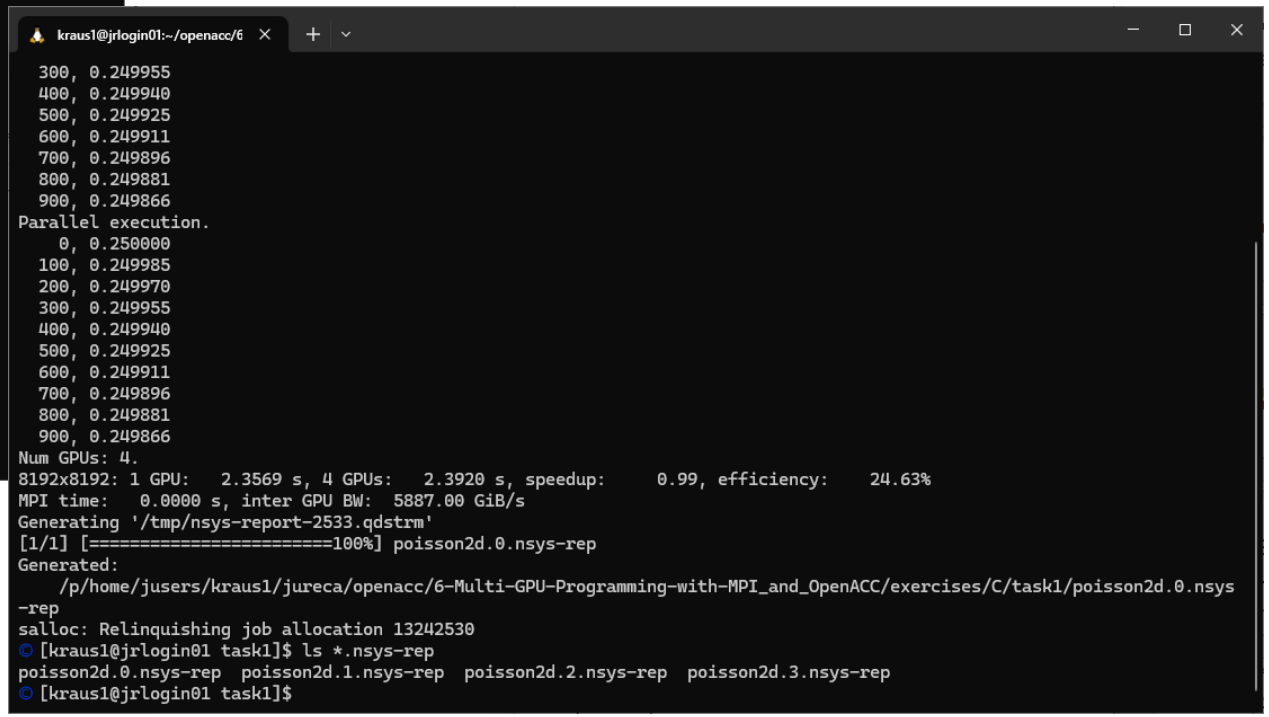
Slrum/JURECA-DC: `SLURM_PROCID`

OpenMPI: `OMPI_COMM_WORLD_RANK`

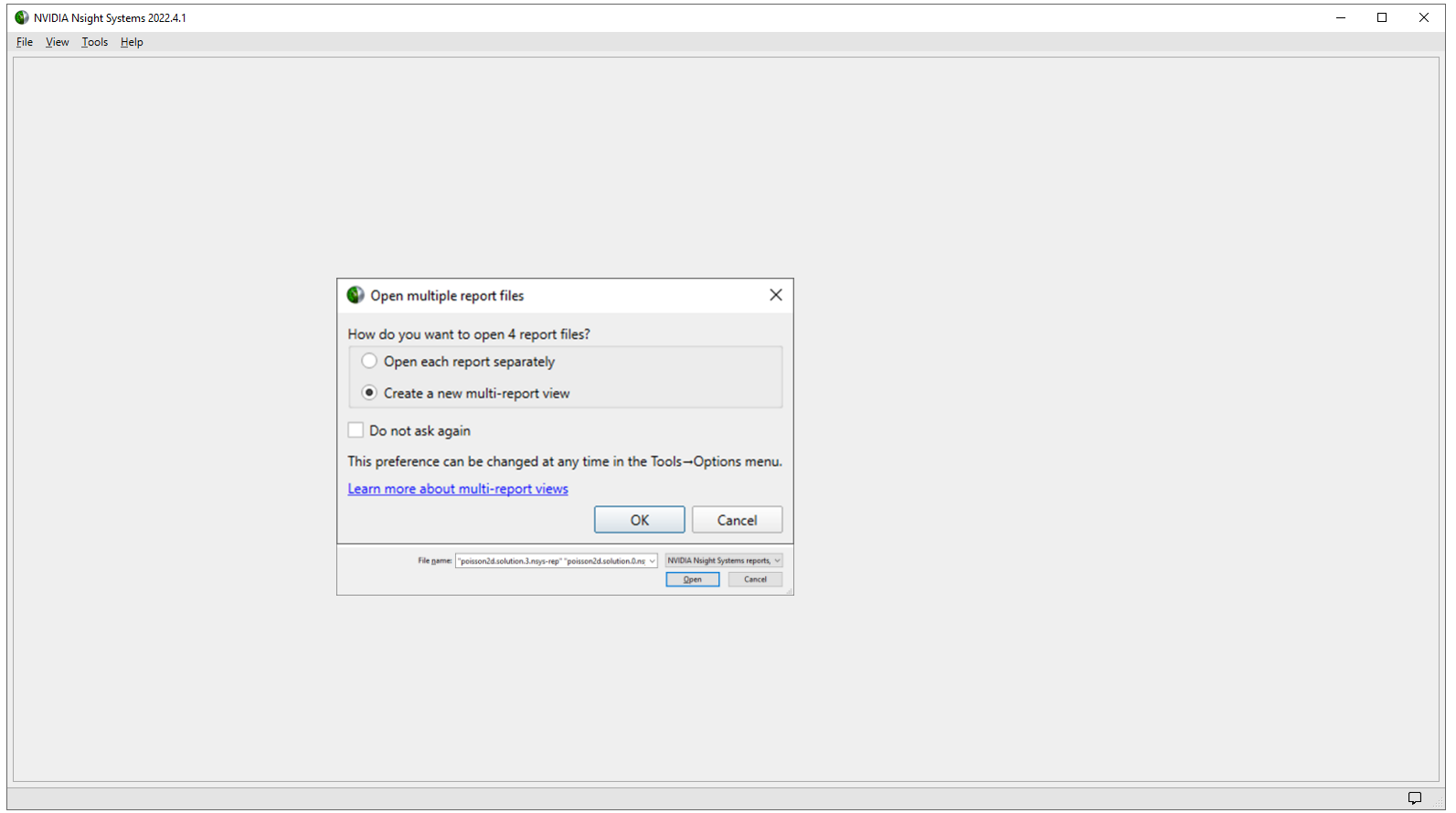
MVAPICH2: `MV2_COMM_WORLD_RANK`

PROFILING MPI+OPENACC APPLICATIONS

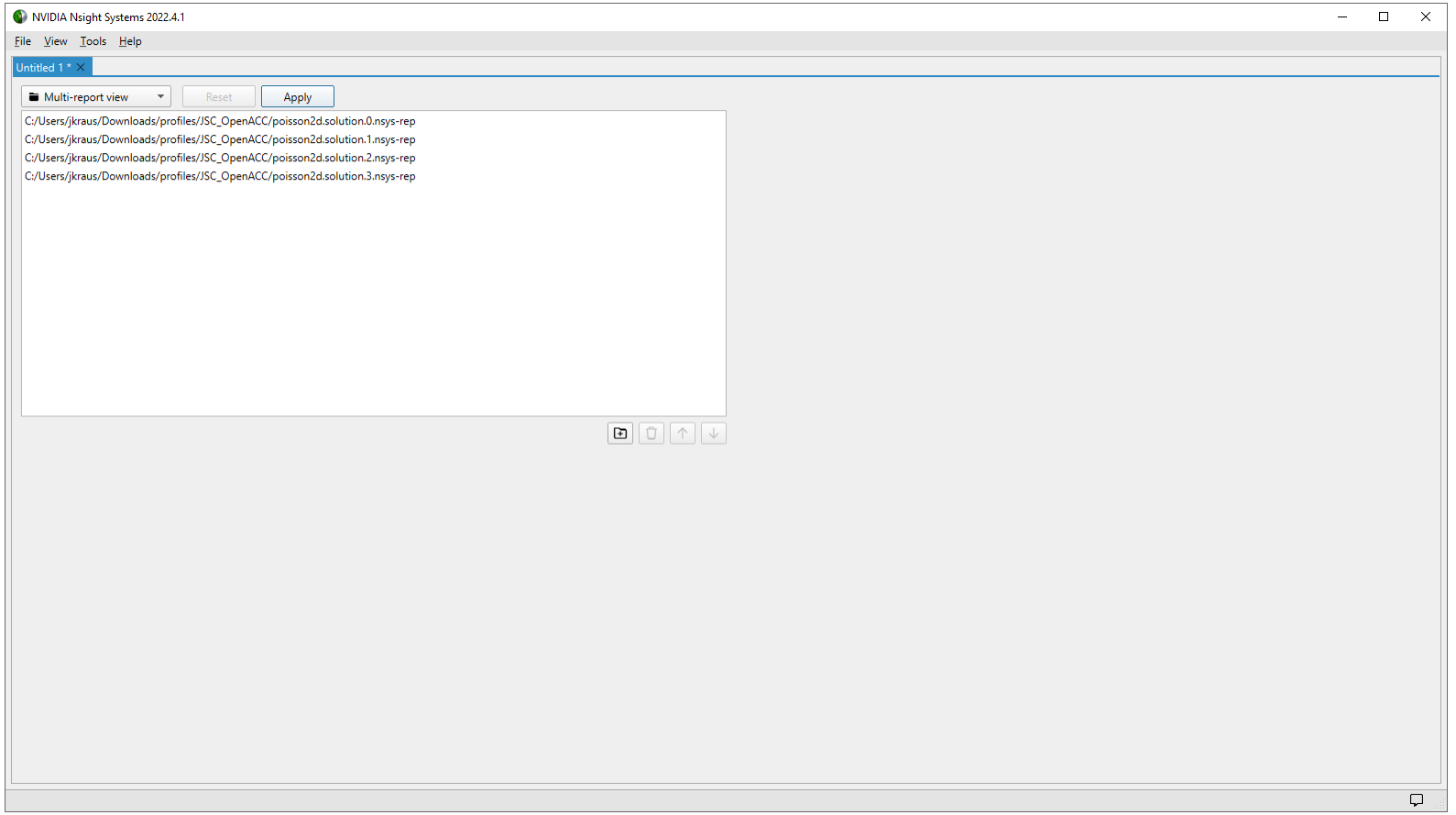




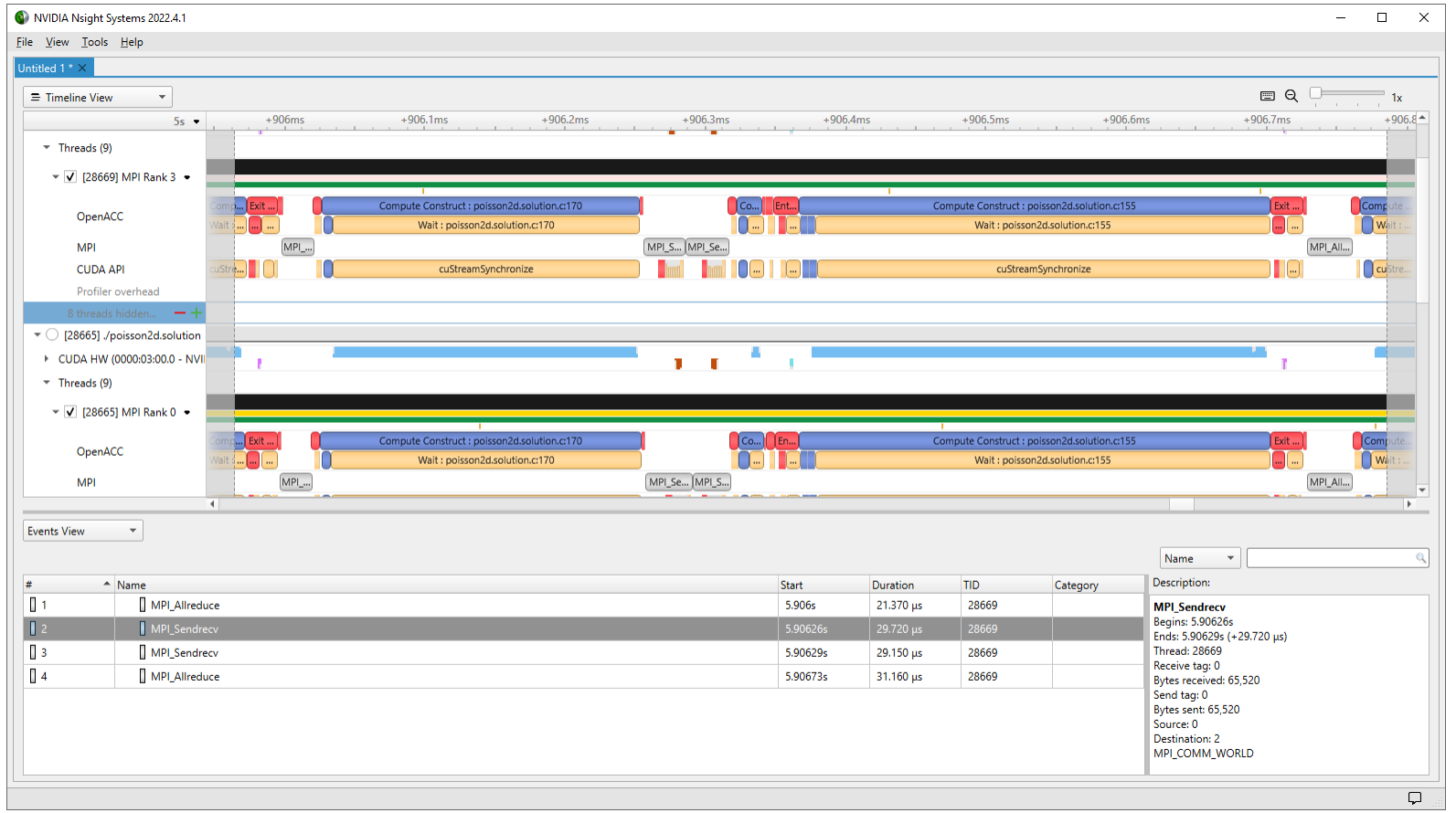
PROFILING MPI+OPENACC APPLICATIONS



PROFILING MPI+OPENACC APPLICATIONS



PROFILING MPI+OPENACC APPLICATIONS

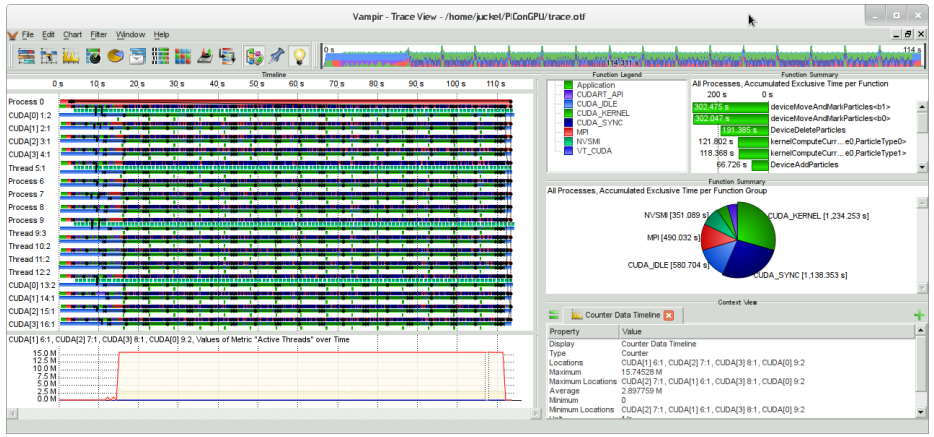


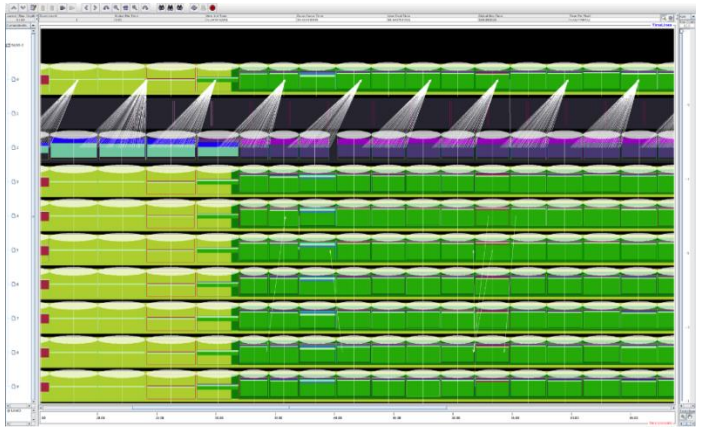
PROFILING MPI+OPENACC APPLICATIONS

Multiple parallel profiling tools are CUDA-aware

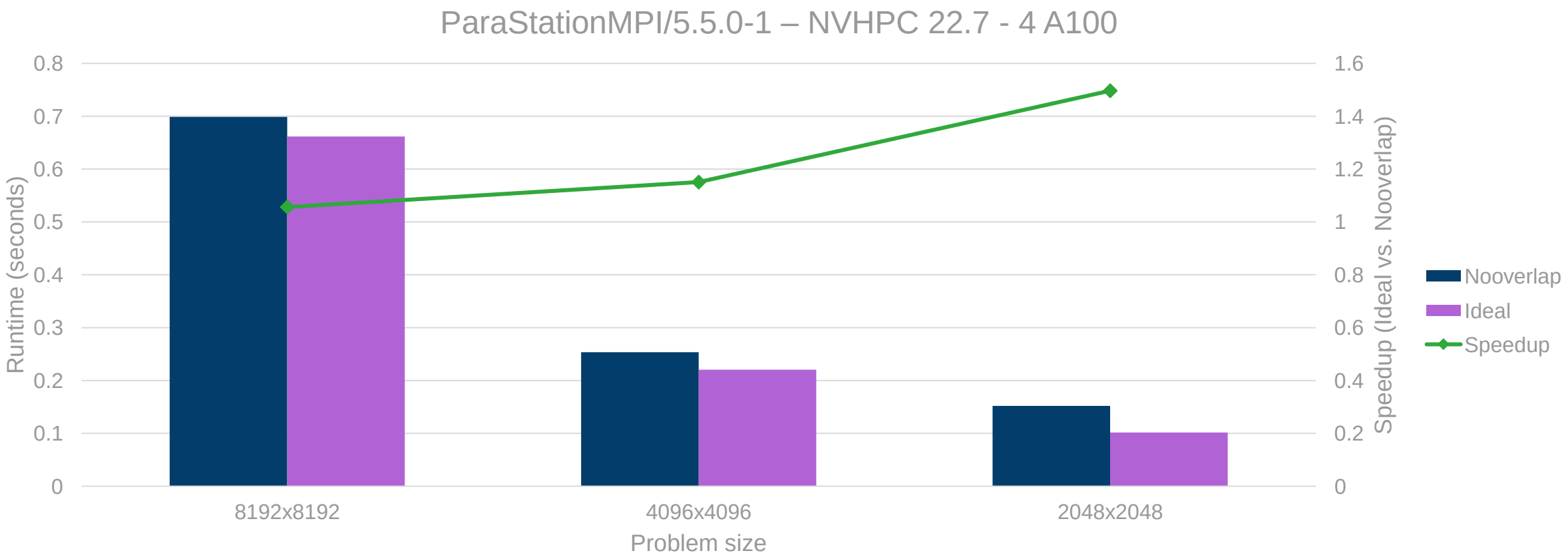
- Score-P
- Vampir
- Tau

These tools are good for discovering MPI issues as well as basic CUDA performance inhibitors.

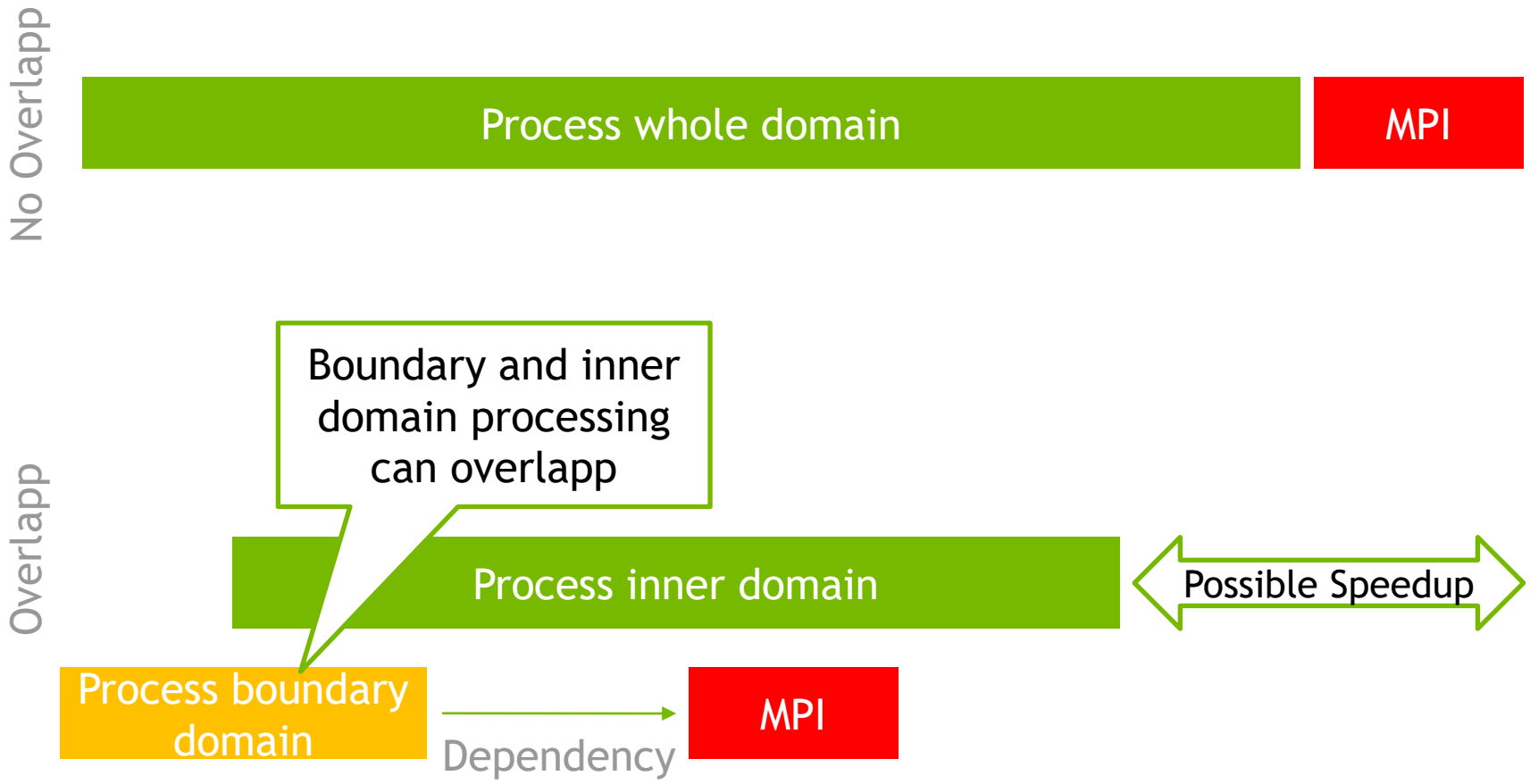




COMMUNICATION + COMPUTATION OVERLAP



COMMUNICATION + COMPUTATION OVERLAP



COMMUNICATION + COMPUTATION OVERLAP

```
#pragma acc parallel loop
for ( ... )
        //Process boundary
#pragma acc parallel loop async
for ( ... )
        //Process inner domain

#pragma acc host_data use_device ( A )
{
  //Exchange halo with top and bottom neighbor
  MPI_Sendrecv( A…);
  //…
}
//wait for iteration to finish
#pragma acc wait
```

SCALABILITY METRICS FOR SUCCESS

- Serial Time: Ts:
  - How long it takes to run the problem with a single process
- Parallel Time: Tp
  - How long it takes to run the problem with multiple processes
- Number of Processes: P
  - The number of Processes operating on the task at hand
- Speedup: $S=Ts/Tp$
  - How much faster is the parallel version vs. serial. (optimal is P)
- Efficiency: $E= S/P$
  - How efficient are the processors used (optimal is 1)

TASK 1: APPLY DOMAIN DECOMPOSITION

- Handle GPU affinity
- Halo Exchange

Look for TODOs

```
$ make
mpicc -c -DUSE_DOUBLE -Minfo=accel -fast -acc=gpu -gpu=cc80 poisson2d.c [...]
srun -p dc-gpu-devel --gres=gpu:4 --time 0:10:00 --pty -n 4 ./poisson2d
Jacobi relaxation Calculation: 8192 x 8192 mesh
[...]
Num GPUs: 4.
8192x8192: 1 GPU:   2.3283 s, 4 GPUs:   2.3494 s, spee
MPI time:   0.0001 s, inter GPU BW:  4760.23 GiB/s
```

Make Targets:

`run:`	run `poisson2d` (default)
`poisson2d:`	build `poisson2d` binary
`profile:`	profile with Nsight Systems
`*.solution:`	same as above with solution (`poisson2d.solution.*`)

TASK 2: HIDE MPI COMMUNICATION TIME

- Start copy loop asynchronously
- Wait for async copy loop after MPI comm. is done

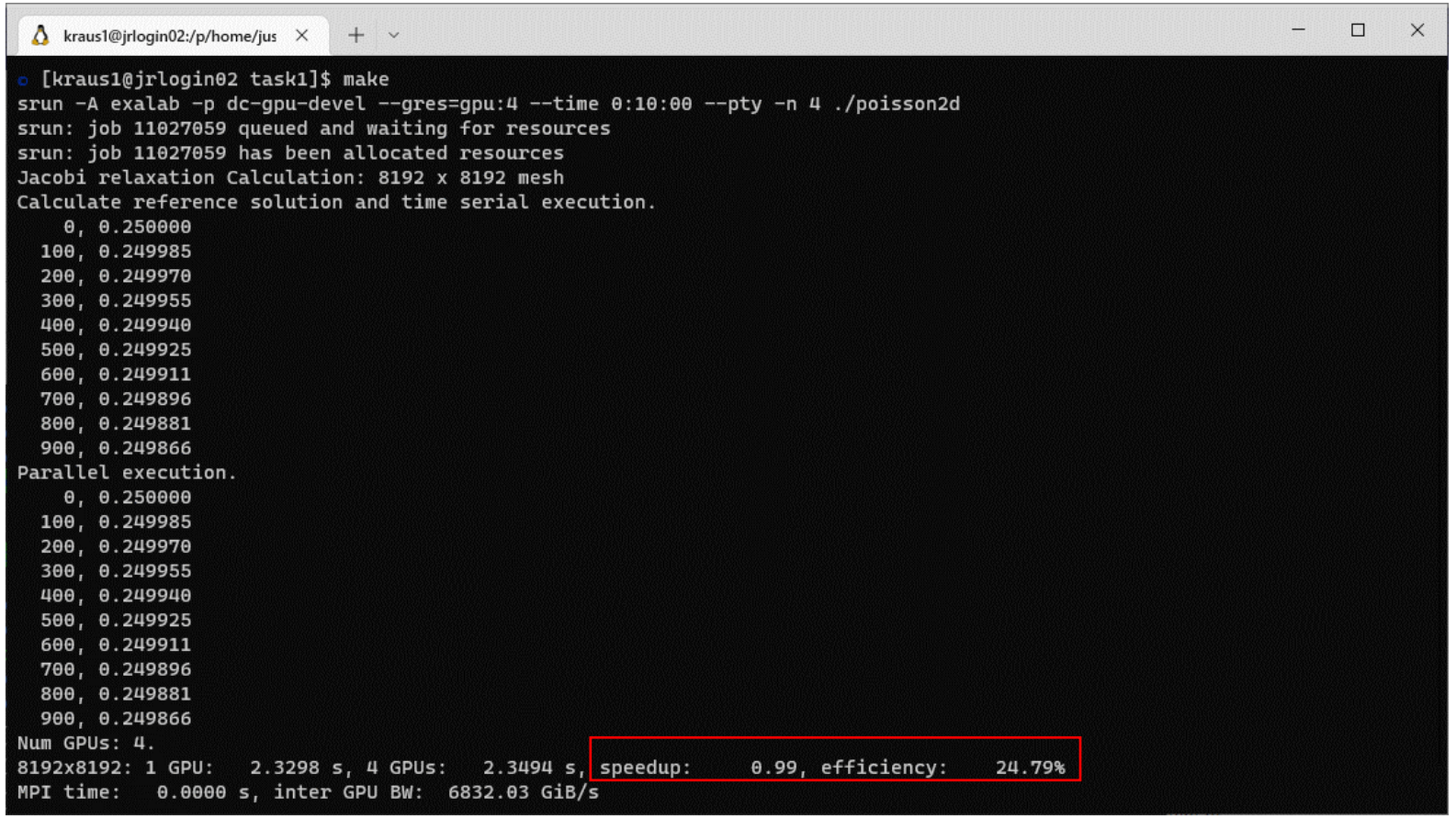
Look for TODOs

```
$ make
mpicc -c -DUSE_DOUBLE -Minfo=accel -fast -acc=gpu -gpu=cc80 poisson2d.c [...]
srun -p dc-gpu-devel --gres=gpu:4 --time 0:10:00 --pty -n 4 ./poisson2d
Jacobi relaxation Calculation: 8192 x 8192 mesh
[...]
Num GPUs: 4.
8192x8192: 1 GPU:   2.3289 s, 4 GPUs:   0.6824 s, spee
MPI time:   0.0373 s, inter GPU BW:     6.54 GiB/s
```

Make Targets:

`run:`	run `poisson2d` (default)
`poisson2d:`	build `poisson2d` binary
`profile:`	profile with Nsight Systems
`*.solution:`	same as above with solution (`poisson2d.solution.*`)

TASK 1: INITIAL VERSION



TASK 1: SOLUTION

```
//Initialize MPI and determine rank and size

MPI_Init(&argc, &argv);

MPI_Comm_rank(MPI_COMM_WORLD, &rank);

MPI_Comm_size(MPI_COMM_WORLD, &size);
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);
}

#pragma acc set device_num( local_rank )

real* restrict const A = (real*) malloc(nx*ny*sizeof(real));
```

`%num_devs` omitted see slide 15

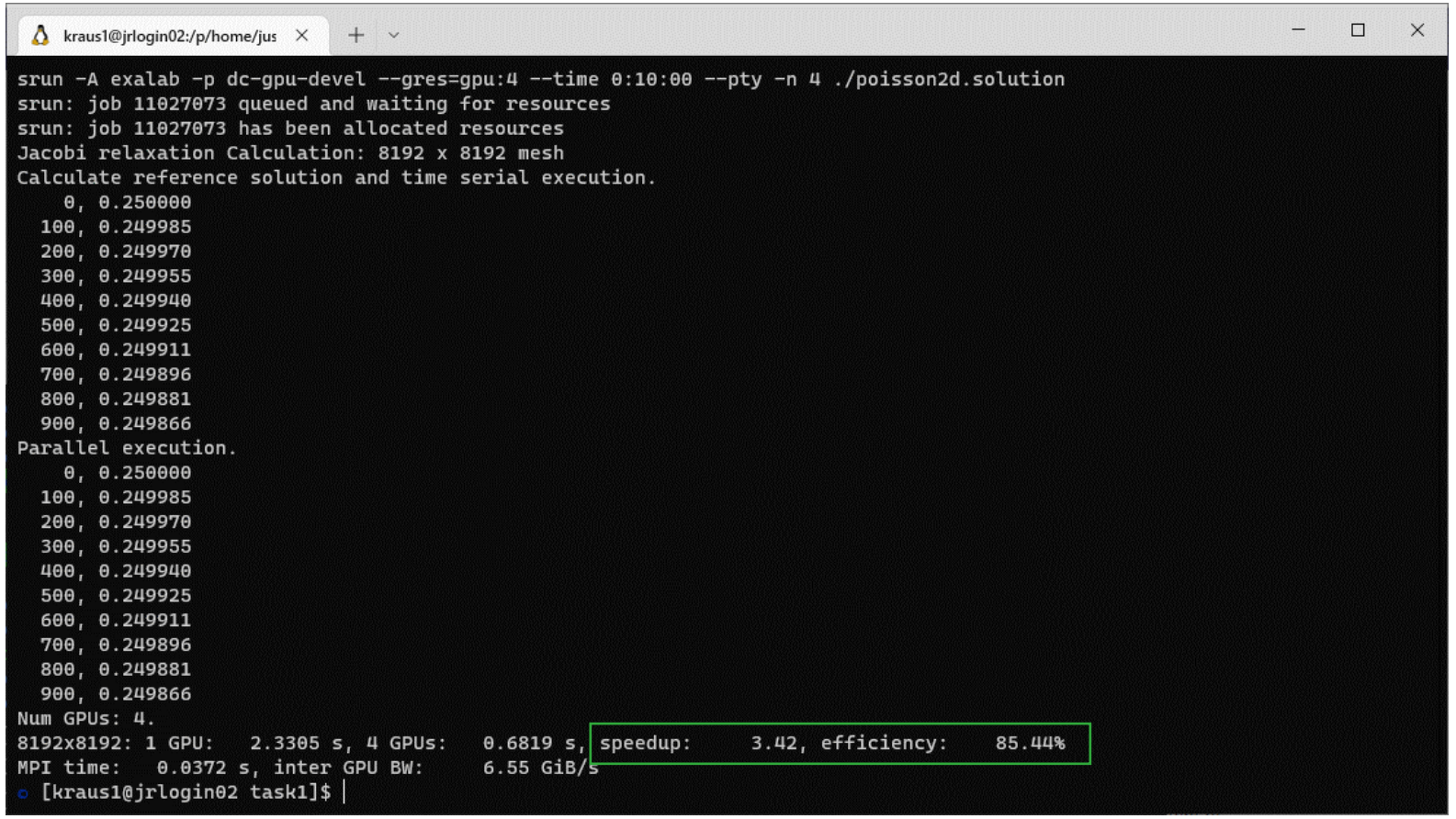
TASK 1: SOLUTION

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

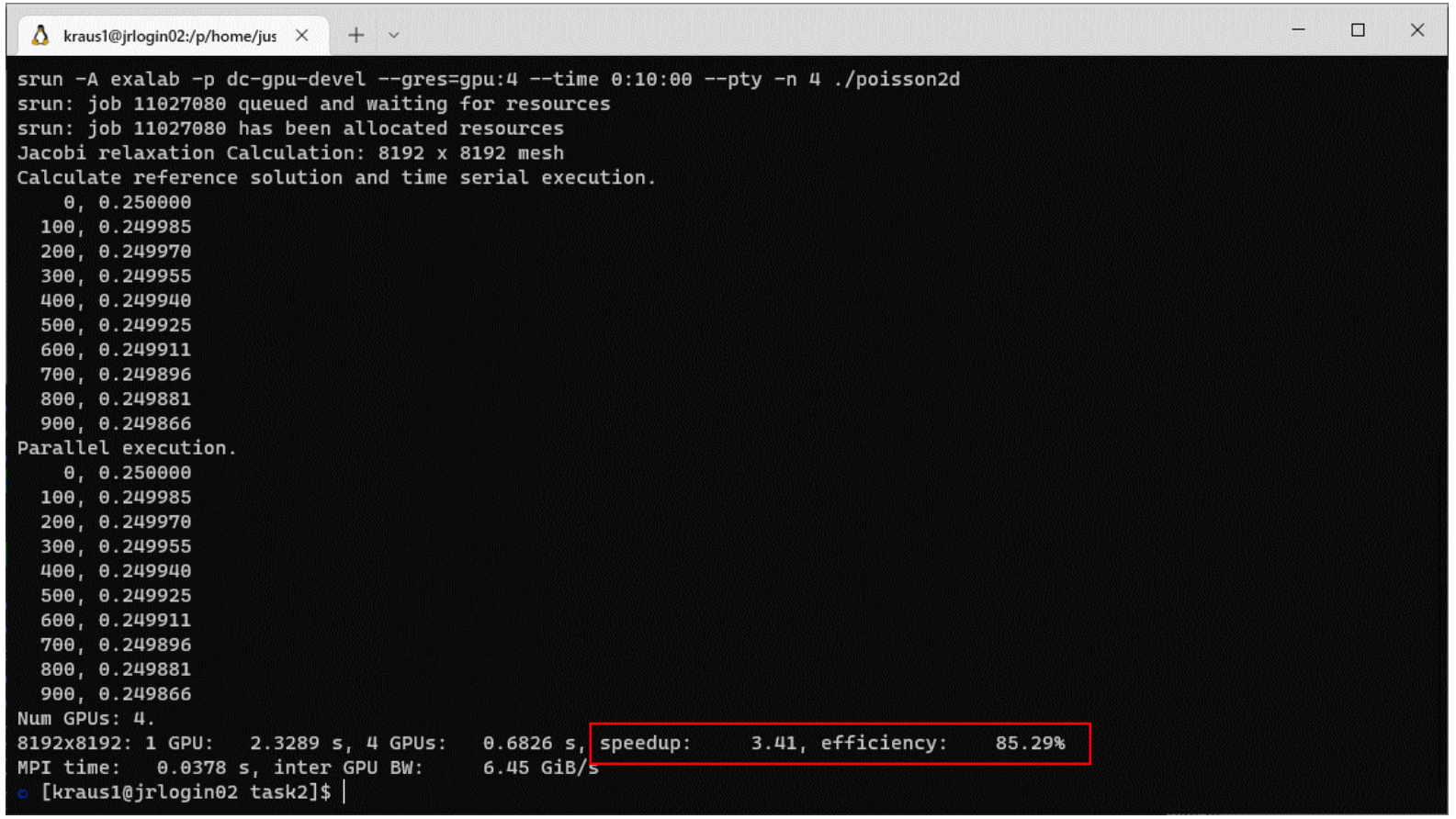
TASK 1: SOLUTION

```
int top = (rank == 0) ? (size-1) : rank-1;
int bottom = (rank == (size-1)) ? 0 : rank+1;
#pragma acc host_data use_device( A ) {
  //1. Sent row iy_start (first modified row) to top receive lower boundary (iy_end)
  //from bottom
  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);
  //2. Sent row (iy_end-1) (last modified row) to bottom
  //receive upper boundary (iy_start-1) from top
  MPI_Sendrecv(A + (iy_end - 1) * nx + ix_start, (ix_end - ix_start), MPI_REAL_TYPE, bottom, 0,
               A + (iy_start - 1) * nx + ix_start, (ix_end - ix_start), MPI_REAL_TYPE, top , 0,
               MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}
```

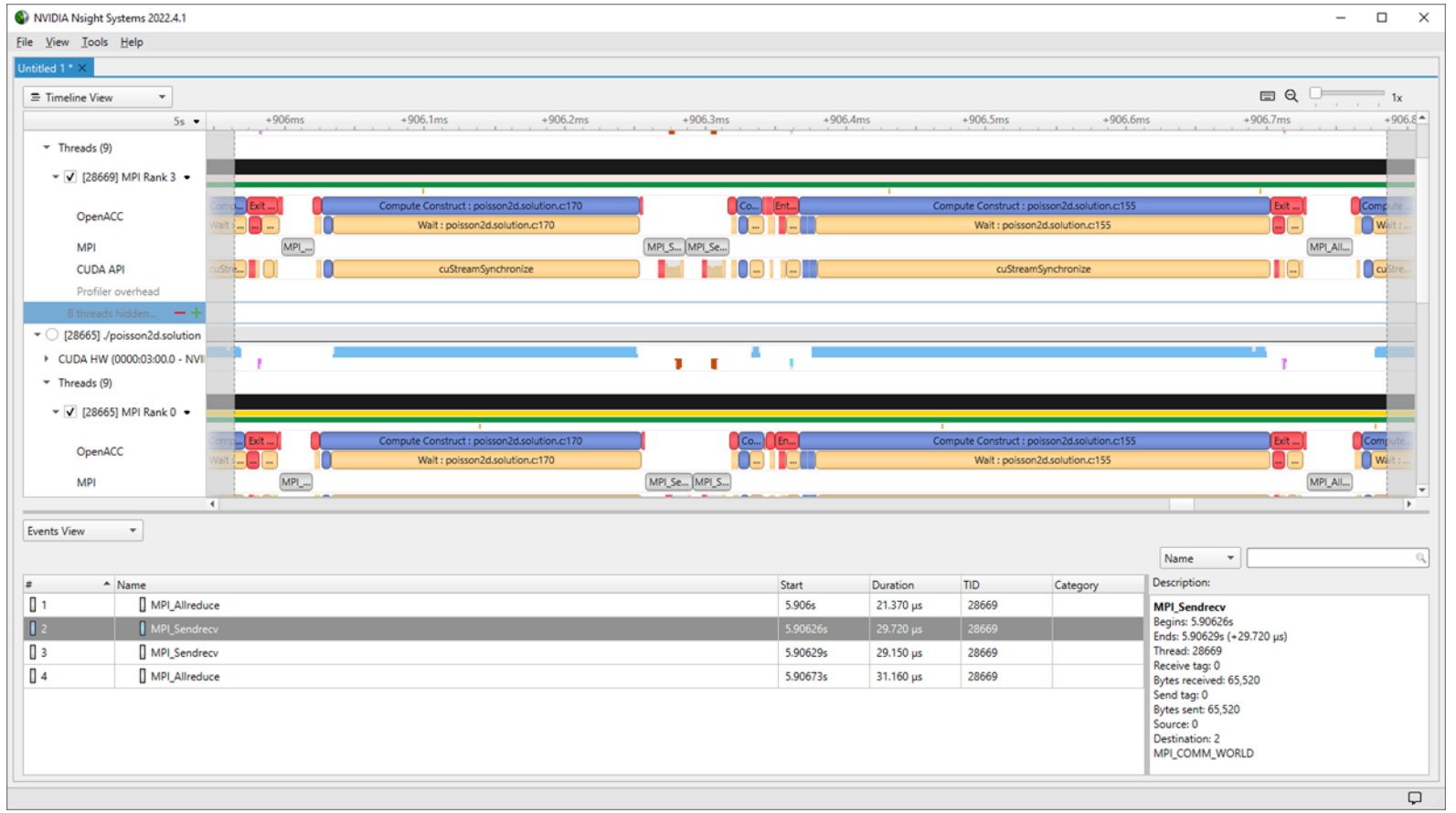
TASK 1: SOLUTION



TASK 2: INITIAL VERSION



TASK 2: INITIAL VERSION

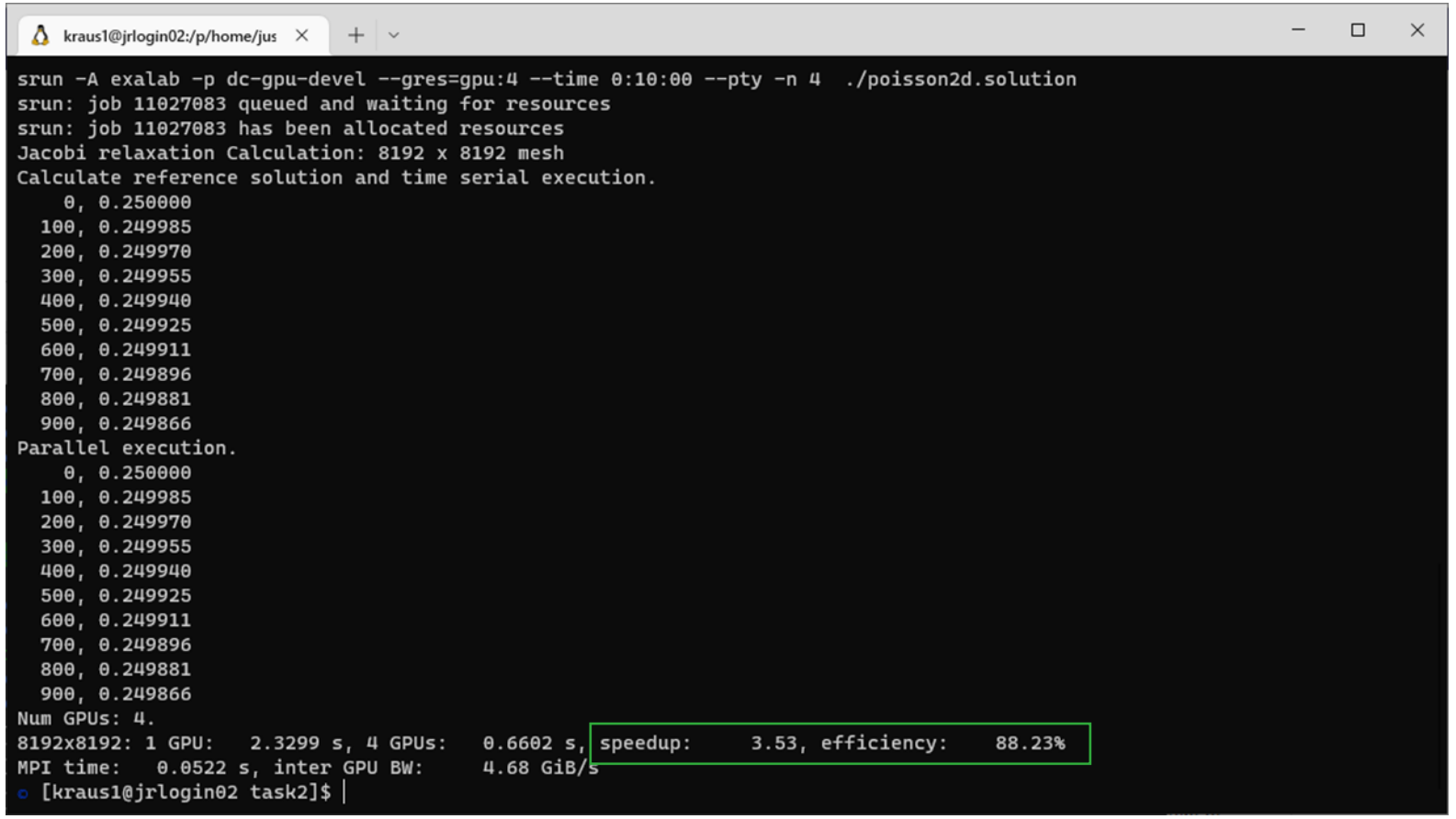


TASK 2: SOLUTION

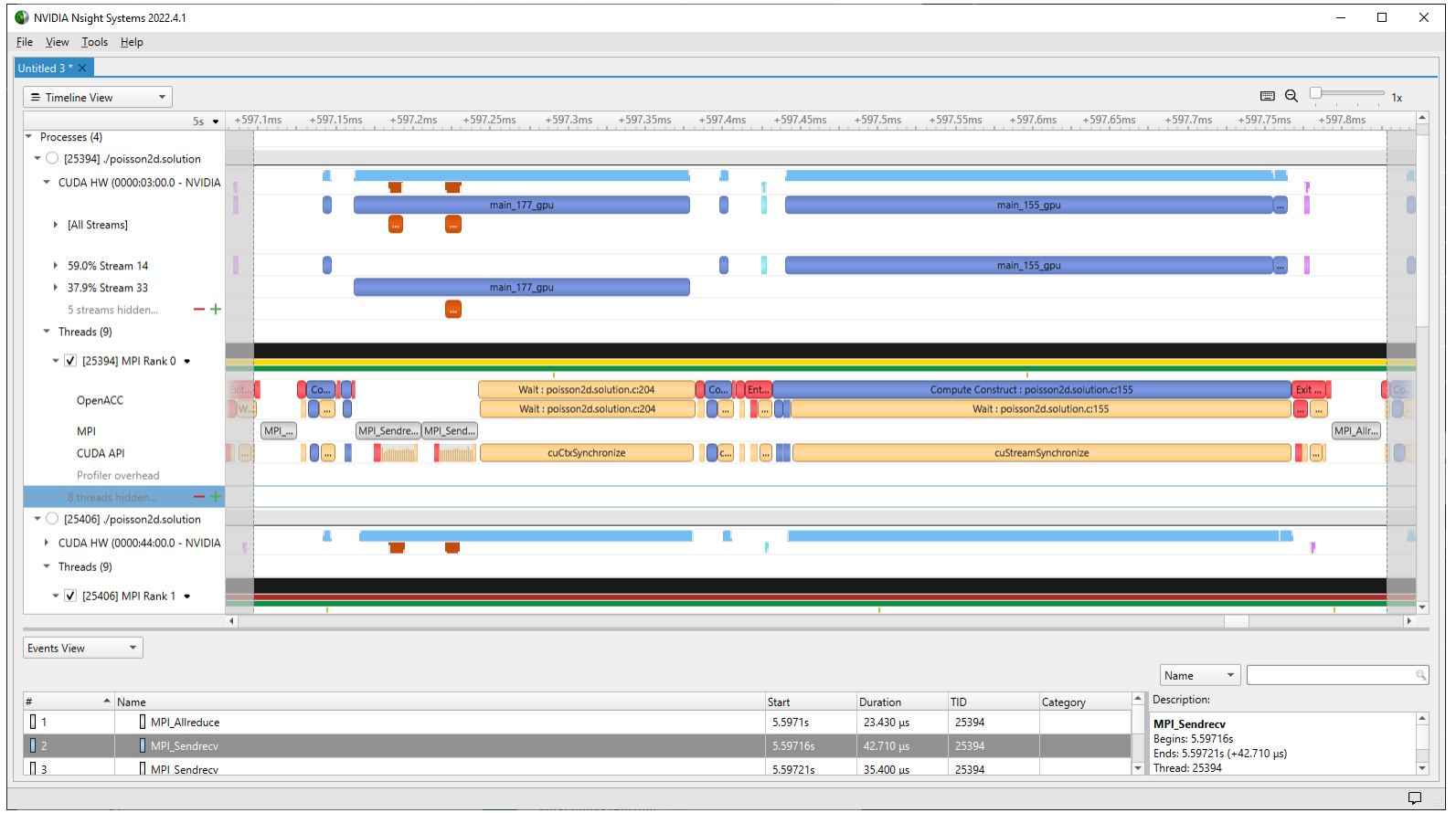
```
#pragma acc parallel loop present(A,Anew)
for( int ix = ix_start; ix < ix_end; ix++ ) {
  A[(iy_start)*nx+ix] = Anew[(iy_start)*nx+ix];
  A[(iy_end-1)*nx+ix] = Anew[(iy_end-1)*nx+ix];
}
#pragma acc parallel loop present(A,Anew) async
for (int iy = iy_start+1; iy < iy_end-1; iy++) {
  for( int ix = ix_start; ix < ix_end; ix++ ) {
    A[iy*nx+ix] = Anew[iy*nx+ix];
  }
}
```

```
int top = (rank == 0) ? (size-1) : rank-1;
int bottom = (rank == (size-1)) ? 0 : rank+1;
#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 );
  MPI_Sendrecv( A+(iy_end-1)*nx+ix_start, (ix_end-ix_start),
                MPI_REAL_TYPE, bottom, 0,
                A+(iy_start-1)*nx+ix_start, (ix_end-ix_start),
                MPI_REAL_TYPE, top , 0,
                MPI_COMM_WORLD, MPI_STATUS_IGNORE );
}
#pragma acc wait
```

TASK 2: SOLUTION



TASK 2: SOLUTION



TASK 2: SOLUTION