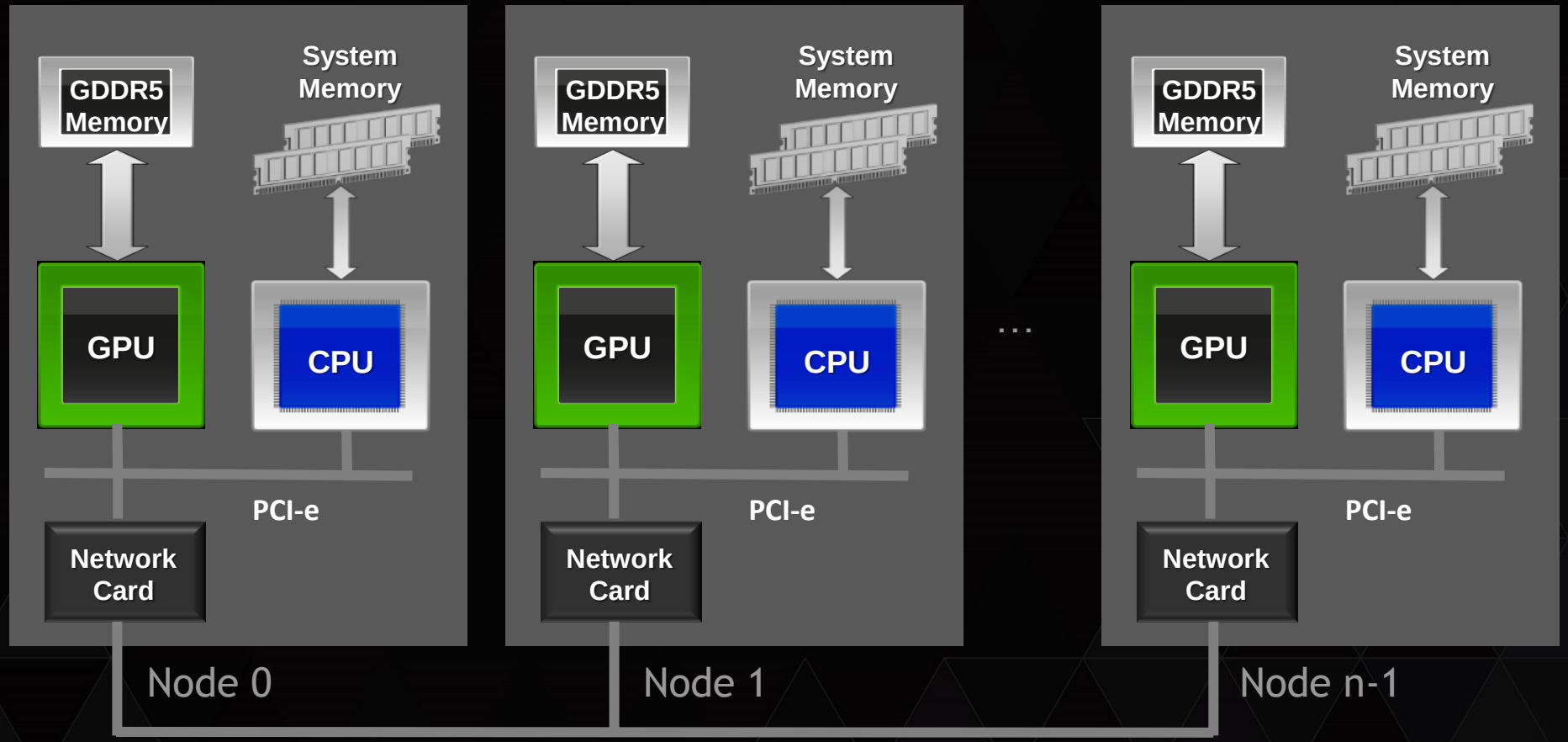
MULTI GPU PROGRAMMING WITH MPI AND OPENACC

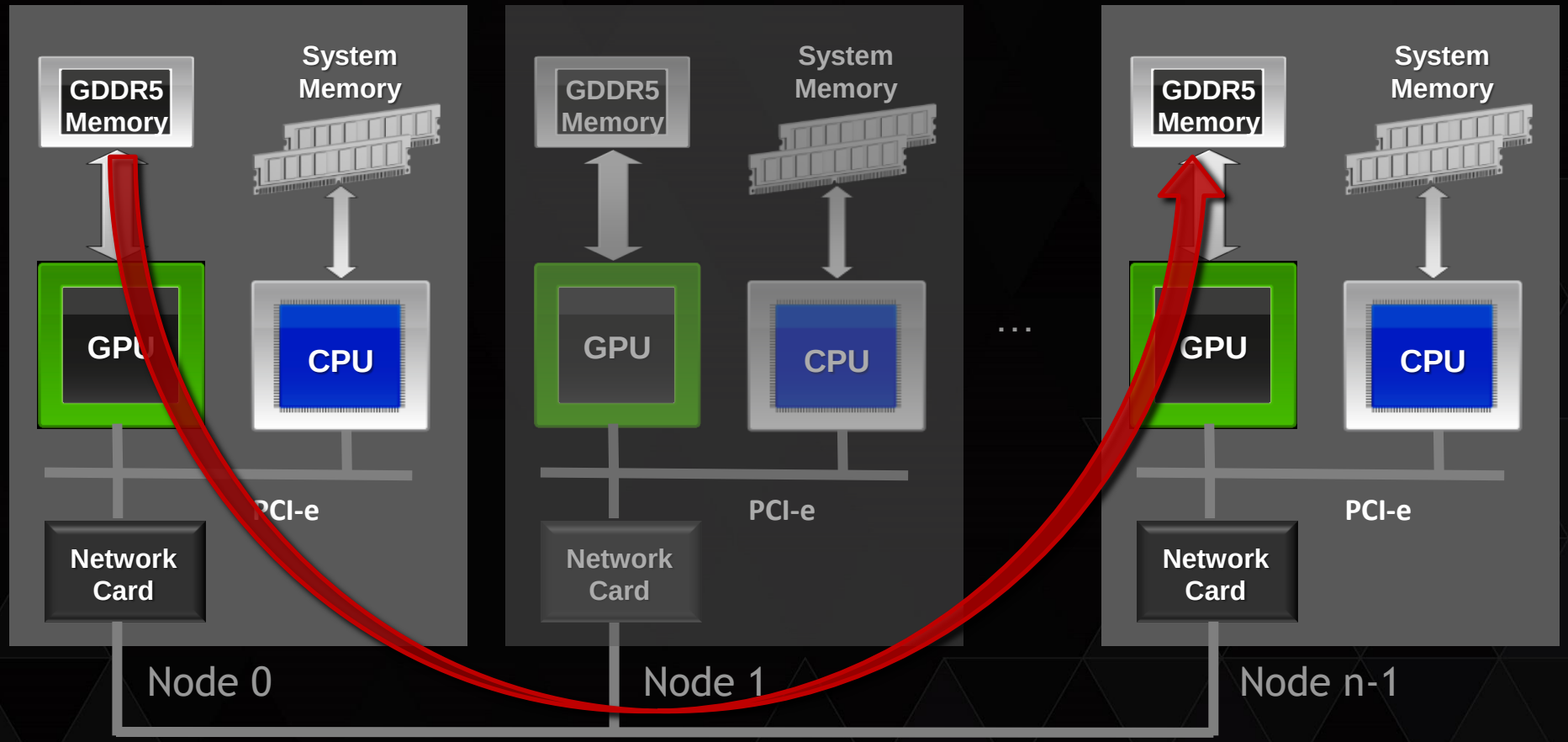
JIRI KRAUS, NVIDIA

MPI+OPENACC



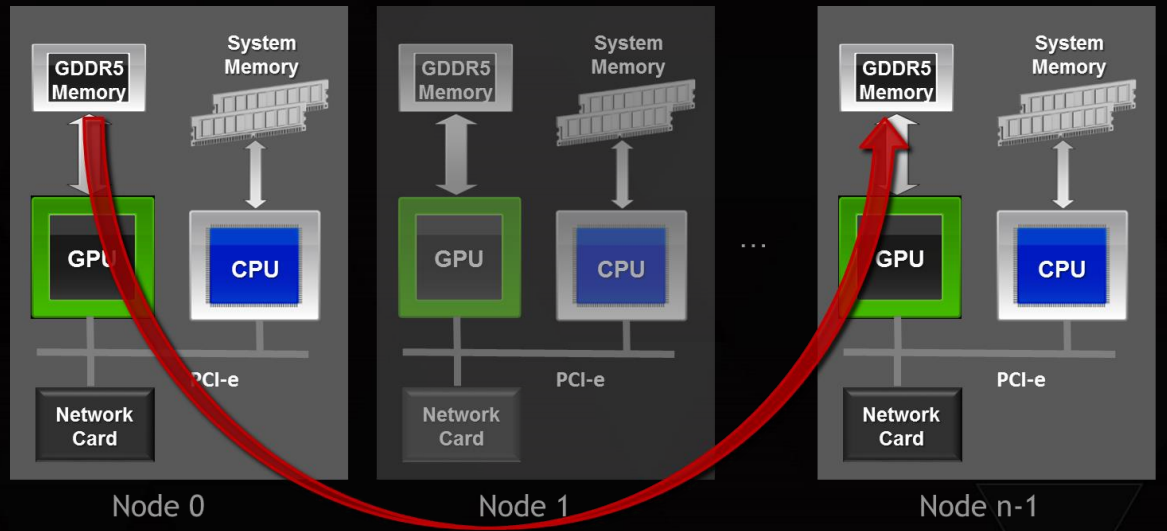
GPU TECHNOLOGY CONFERENCE

MPI+OPENACC



GPU TECHNOLOGY CONFERENCE

MPI+OPENACC




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

//MPI rank n-1
MPI_Recv(r_buf_d,size,MPI_CHAR,0,tag,MPI_COMM_WORLD,&stat);
```

GPU TECHNOLOGY CONFERENCE

WHAT YOU WILL LEARN

- What MPI and OpenACC is
- How to use MPI for inter GPU communication with OpenACC
- How to use the NVIDIA profiler for MPI+OpenACC applications
- How to hide MPI communication times

GPU TECHNOLOGY CONFERENCE

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, MVAPICH, IBM Platform MPI, Cray MPT, …

GPU TECHNOLOGY CONFERENCE

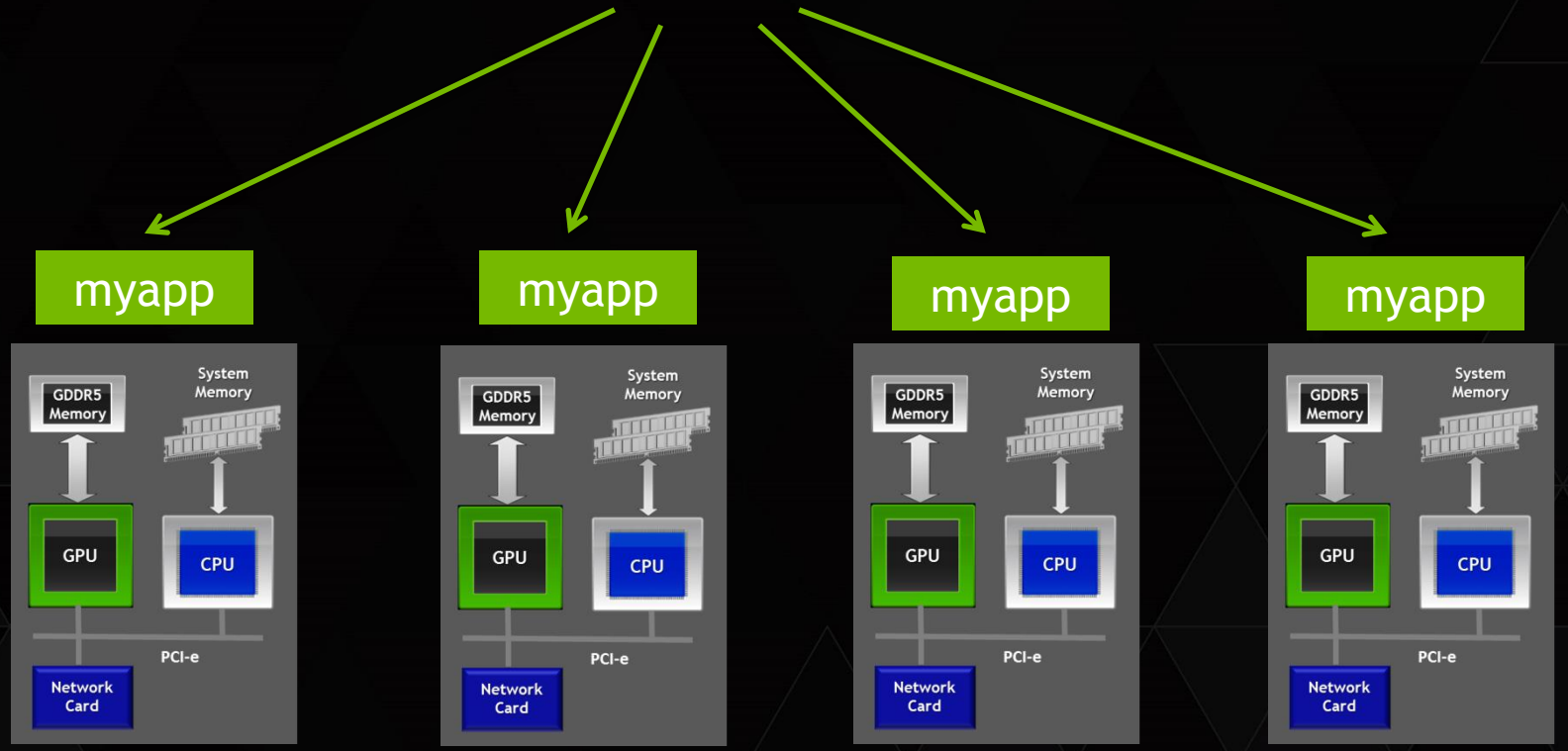
MPI - A MINIMAL PROGRAM

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

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

MPI - COMPILING AND LAUNCHING

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



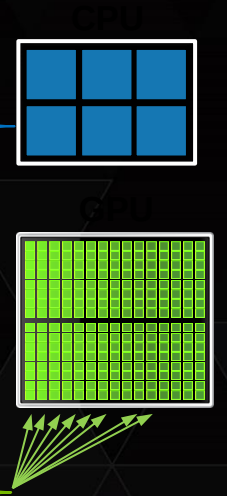
GPU TECHNOLOGY CONFERENCE

OPENACC

- Simple Compiler hints
- Compiler Parallelizes code
- Works on many-core GPUs & multicore CPUs

OpenACC Compiler Hint

```
while ( error>tol && iter<iter_max )
{
    error = 0.f;
#pragma acc kernels
    for( int j = 1; j < N-1; j++)
    {
        for( int i = 1; i < M-1; i++ )
        {
            //...
        }
    }
//...
```



OPENACC – 2 BASIC STEPS

- Step 1: Annotate source code with directives:

```
#pragma acc kernels
    for( int j = 1; j < N-1; j++)
    {
```

- Step 2: Compile & run:

```
pgcc -acc -ta=nvidia laplace2d.c -o laplace2d
```

GPU TECHNOLOGY CONFERENCE

OPENACC

Copy arrays into GPU memory

Parallelize code inside region

End of parallel region: Synchronize

End of data region: Copy data back

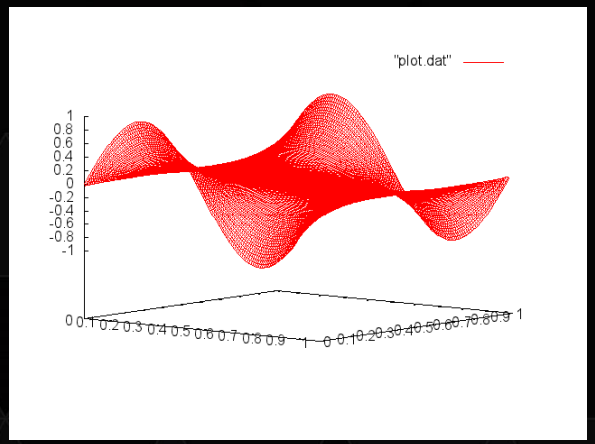
```
#pragma acc data copy(A,Anew)
while ( error > tol && iter < iter_max )
{
    error = 0.f;
#pragma acc kernels
    for( int j = 1; j < N-1; j++)
    {
        for( int i = 1; i < M-1; i++ )
        {
            Anew[j][i]=0.25f*(A[j][i+1]+A[j][i-1]
                            + A[j-1][i]+A[j+1][i]);
            error=fmaxf(error,fabsf(Anew[j][i]-A[j][i]));
        }
    }
//...
}
```

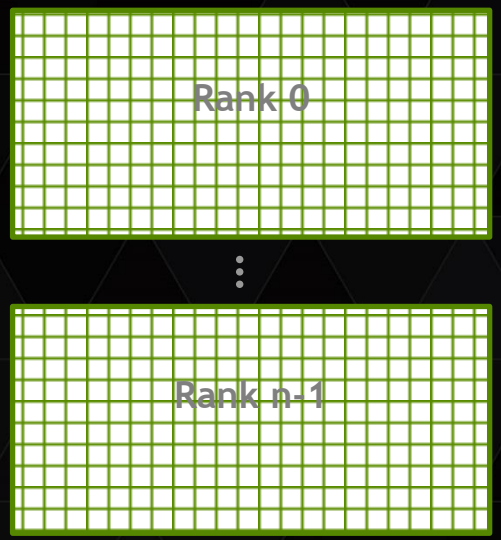
EXAMPLE: JACOBI SOLVER

- Solves the 2D-Laplace equation on a rectangle

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

  - Dirichlet boundary conditions (constant values on boundaries) on left and right boundary
  - Periodic boundary conditions Top Bottom
- 1D domain decomposition with n domains





GPU TECHNOLOGY CONFERENCE

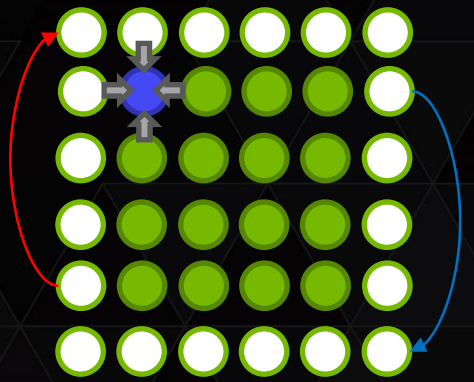
EXAMPLE: JACOBI SOLVER - SINGLE GPU

While not converged

- Do Jacobi step:

```
for (int i=1; i < n-1; i++)
  for (int j=1; j < m-1; j++)
    Anew[i][j] = 0.0f - 0.25f*(A[i-1][j] + A[i+1][j]
                              +A[i][j-1] + A[i][j+1])
```

- Copy `Anew` to `A`
- Apply periodic boundary conditions



- Next iteration

GPU TECHNOLOGY CONFERENCE

HANDLING GPU AFFINITY

- Rely on process placement (with one rank per GPU)*

```
int rank = 0;
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
int ngpus = acc_get_num_devices(acc_device_nvidia); // ngpus == ranks per node
int devicenum = rank % ngpus;
acc_set_device_num(devicenum,acc_device_nvidia);
```

GPU TECHNOLOGY CONFERENCE

*This assumes the node is homogeneous, i.e. that all the GPUs are the same. If you have different GPUs in the same node then you may need some more complex GPU selection

CONNECTION INSTRUCTIONS

- Navigate to nvlabs.qwiklab.com
- Login or create a new account
- Select the "Instructor-Led Hands-on Labs" class
- Find the lab called "Multi GPU Programming with MPI and OpenACC (S5711 - GTC 2015)" and click Start
- After a short wait, lab instance connection information will be shown
- Please ask Lab Assistants for help!

GPU TECHNOLOGY CONFERENCE

https://github.com/jirikraus/Multi_GPU_Programming_with_MPI_and_OpenACC

TASK1: ADD MPI BOILER PLATE CODE

- Log into cloud node
- TODOs in `task1/laplace2d.c` and `task1/Makefile`
  - Use MPI compiler wrapper (`mpicc`)
  - Start with MPI launcher (`mpirun -np …`)
  - Include MPI header (`mpi.h`)
  - Initialize MPI (`MPI_Init, MPI_Comm_rank, MPI_Comm_size`)
  - Handle GPU Affinity
  - Insert barriers to ensure correct timing (`MPI_Barrier`)
  - Finalize MPI (`MPI_Finalize`)
  - Compile and run: `make`

https://www.open-mpi.org/doc/v1.8

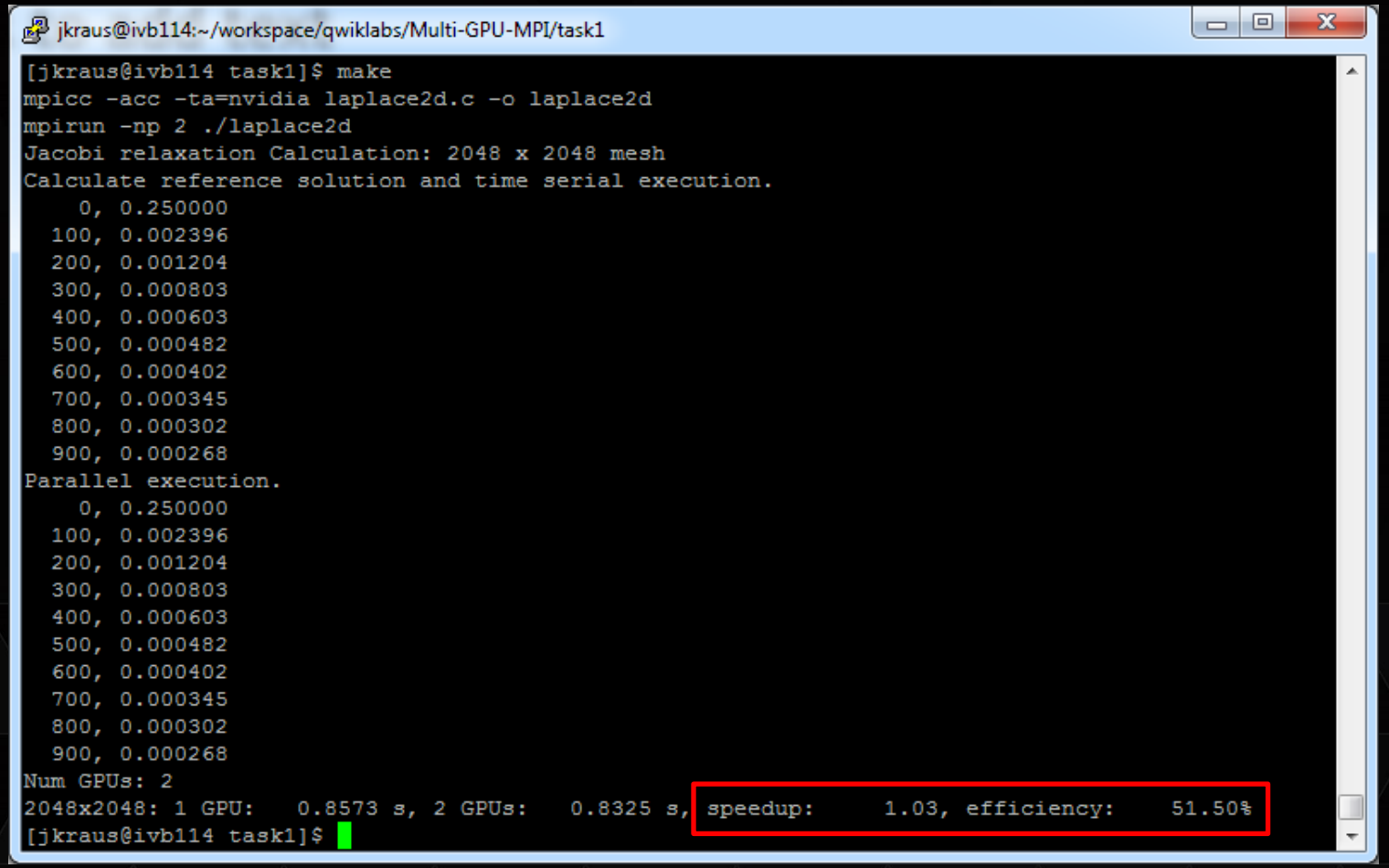
GPU TECHNOLOGY CONFERENCE

SCALABILITY METRICS FOR SUCCESS

- Serial Time: T_s:
  - How long it takes to run the problem with a single process
- Parallel Time: T_p
  - 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: $S = \frac{T_s}{T_p}$
  - How much faster is the parallel version vs. serial. (optimal is P)
- Efficiency: $E = \frac{S}{P}$
  - How efficient are the processors used (optimal is 1)

GPU TECHNOLOGY CONFERENCE

TASK1: RESULTS



DOMAIN DECOMPOSITION

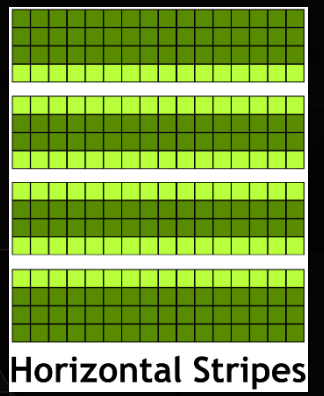
Different Ways to split the work between processes:

Minimizes number of neighbors:

- Communicate to less neighbors
- Optimal for latency bound communication

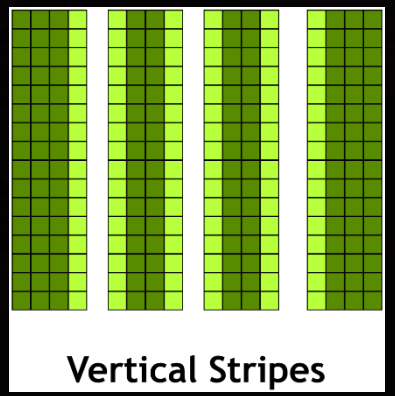
Minimizes surface area/volume ratio:

- Communicate less data
- Optimal for bandwidth bound communication



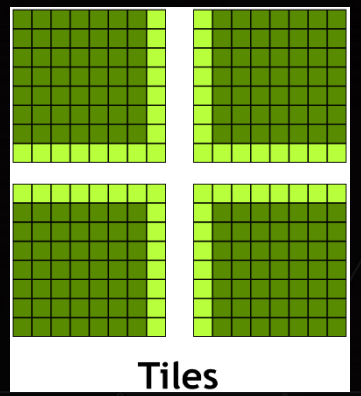
Horizontal Stripes

Contiguous if data is row-major



Vertical Stripes

Contiguous if data is column-major



Tiles

EXAMPLE: JACOBI SOLVER - MULTI GPU

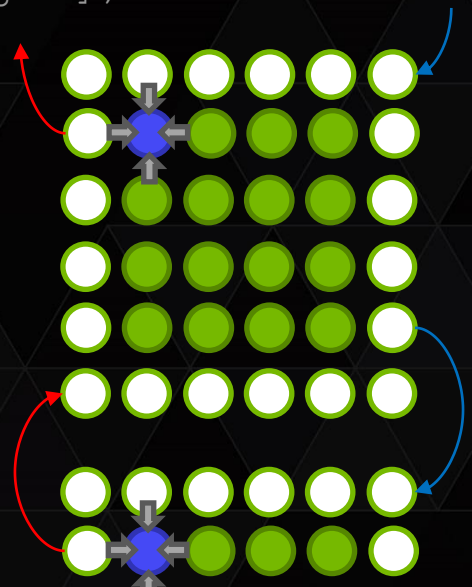
While not converged

- Do Jacobi step:

```
for (int i=1; i < n-1; i++)
  for (int j=1; j < m-1; j++)
    Anew[i][j] = 0.0f - 0.25f*(A[i-1][j] + A[i+1][j]
                              +A[i][j-1] + A[i][j+1])
```

- Copy `Anew` to `A`
- Apply periodic boundary conditions
- Exchange halo with 1 to 2 neighbors
- Next iteration

One-step with ring exchange



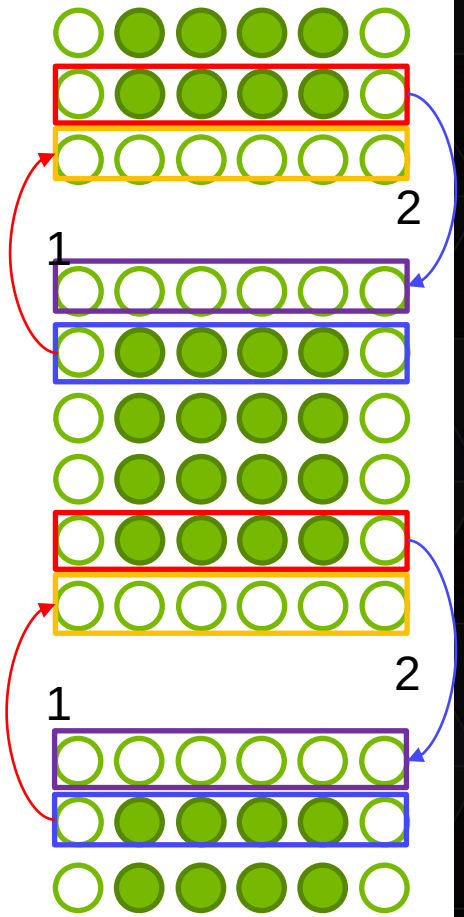
GPU TECHNOLOGY CONFERENCE

EXAMPLE: JACOBI - TOP/BOTTOM HALO

OpenACC

```
#pragma acc host_data use_device ( A ) {
MPI_Sendrecv(A[jstart], M, MPI_FLOAT, top, 0,
             A[jend], M, MPI_FLOAT, bottom, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);

MPI_Sendrecv(A[(jend-1)], M, MPI_FLOAT, bottom, 0,
             A[(jstart-1)], M, MPI_FLOAT, top, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}
```



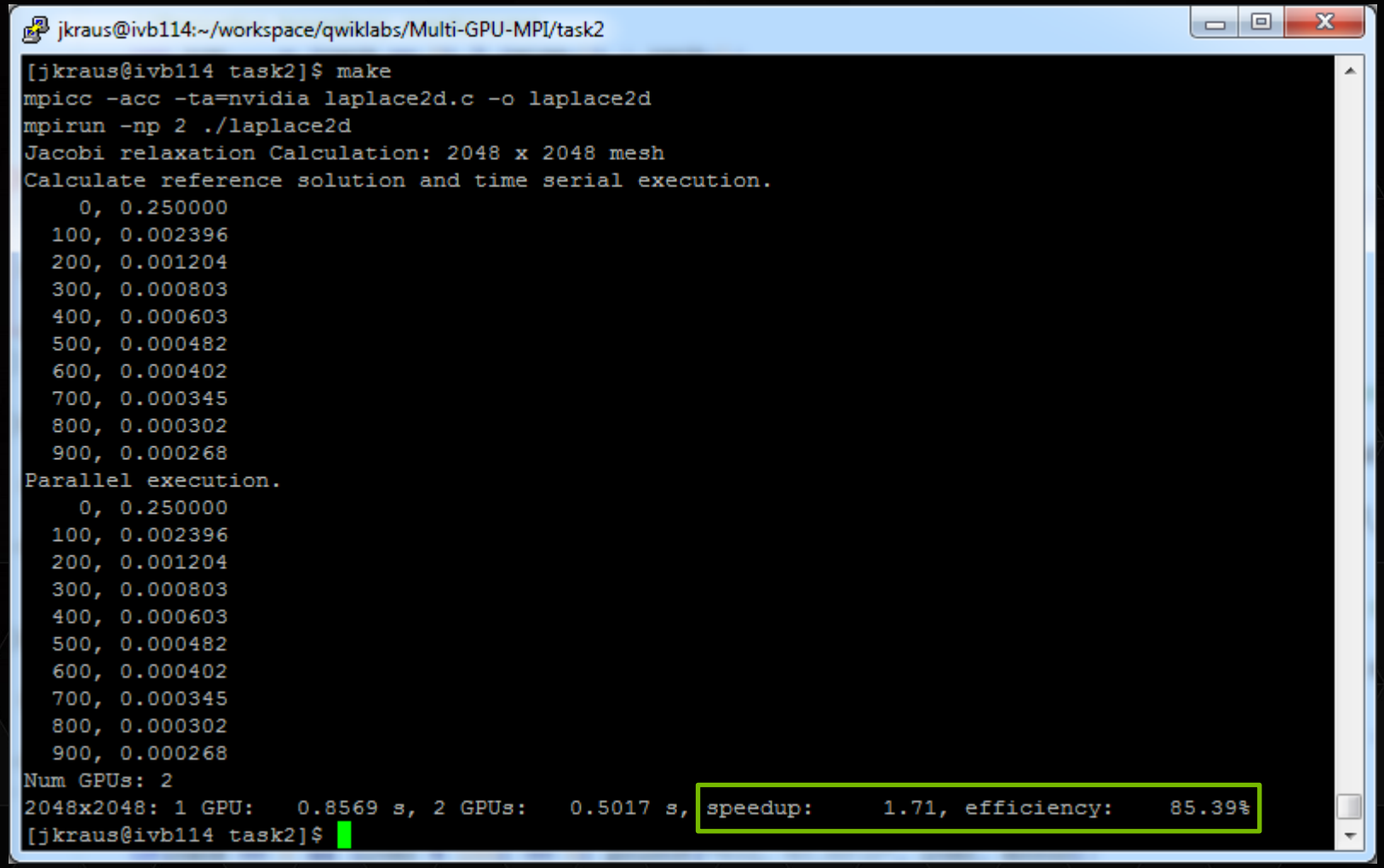
TASK2: DISTRIBUTE WORK ACROSS GPUS

- TODOs in `task2/laplace2d.c`
  - Distribute work across GPUs (`jstart,jend`)
  - Calculate global error (`MPI_Allreduce`)
  - Handle periodic boundary and communicate domain boundaries with MPI (`MPI_Sendrecv`)

https://www.open-mpi.org/doc/v1.8

GPU TECHNOLOGY CONFERENCE

TASK2: RESULTS



PROFILING MPI+OPENACC APPLICATIONS

Using nvprof+NVVP:

- Embed MPI Rank in output filename to be read by NVVP

```
mpirun -np 2 nvprof --output-profile
profile.%q{OMPI_COMM_WORLD_RANK}.out …
```

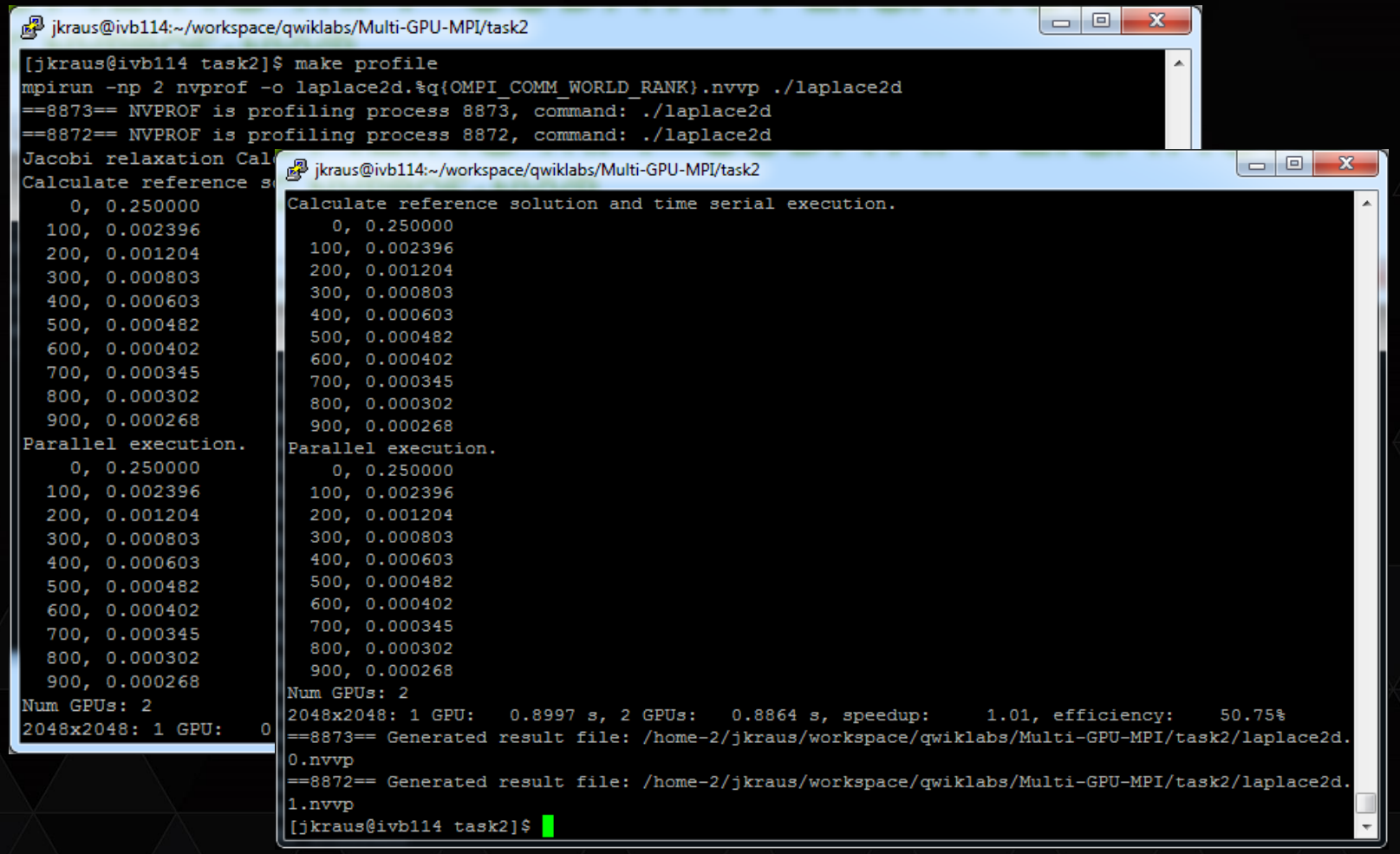
Using nvprof only:

- Only save the textual output

```
mpirun -np 2 nvprof --log-file profile
.%q{OMPI_COMM_WORLD_RANK}.log
```

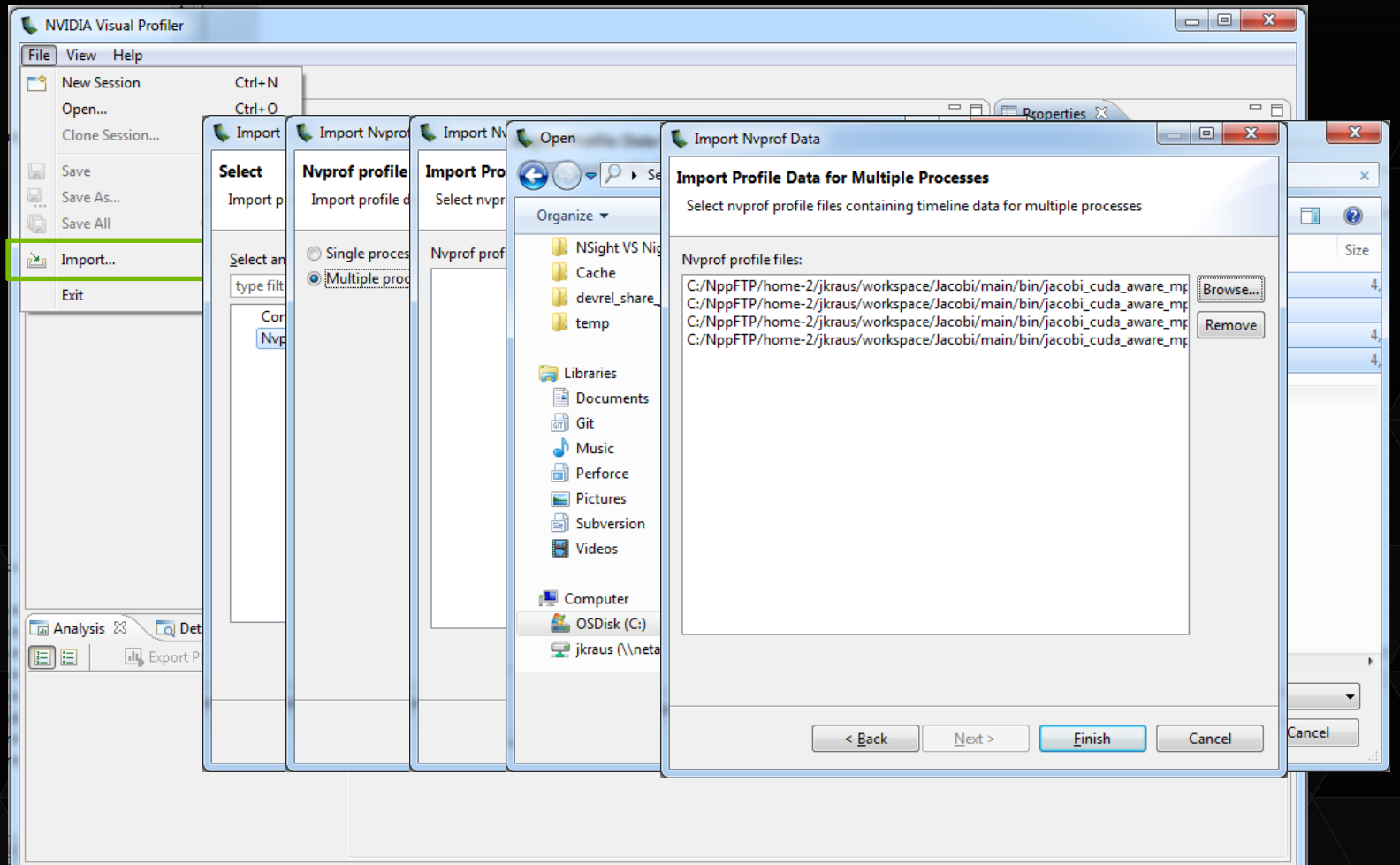
GPU TECHNOLOGY CONFERENCE

PROFILING MPI+OPENACC APPLICATIONS



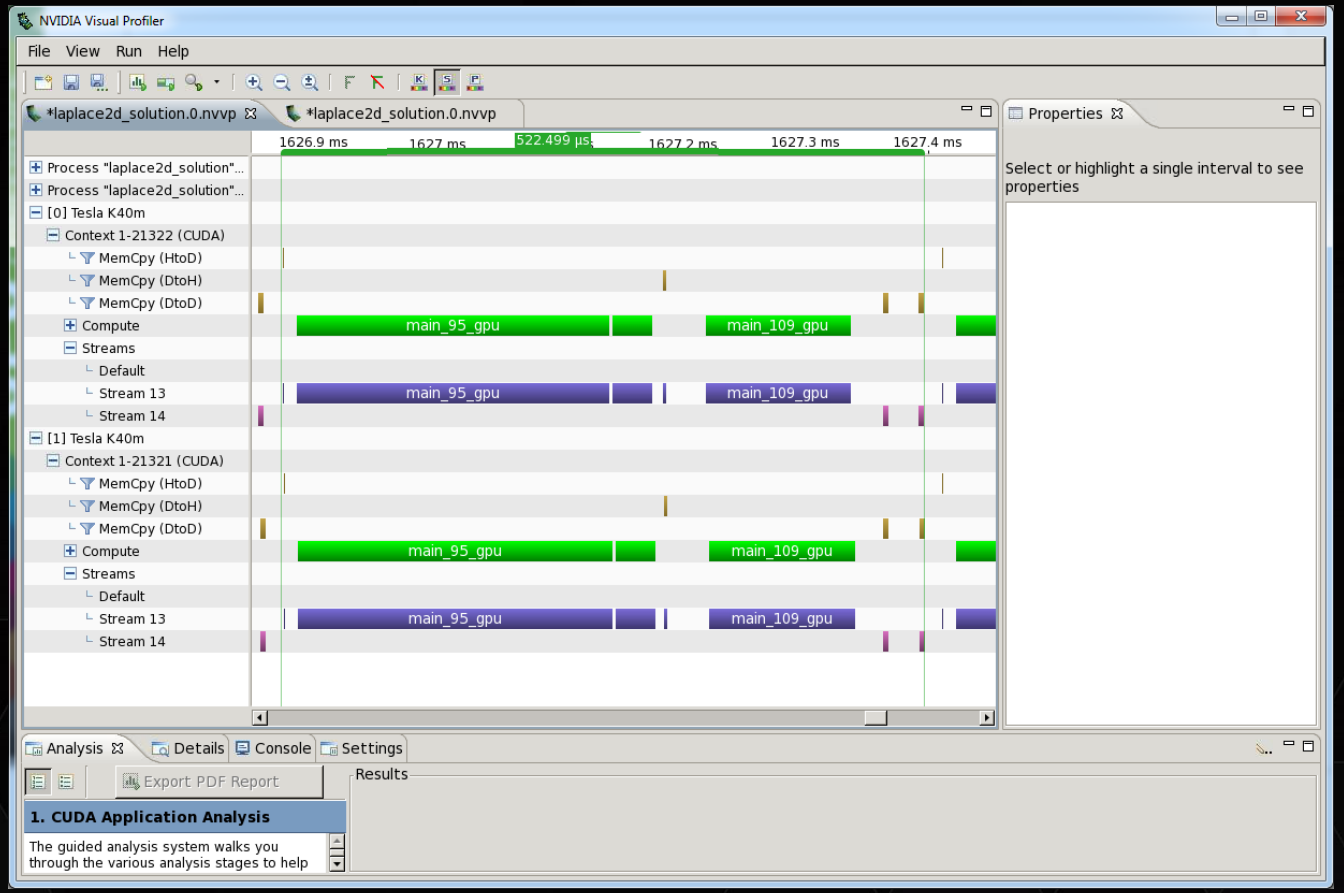
GPU TECHNOLOGY CONFERENCE

PROFILING MPI+OPENACC APPLICATIONS

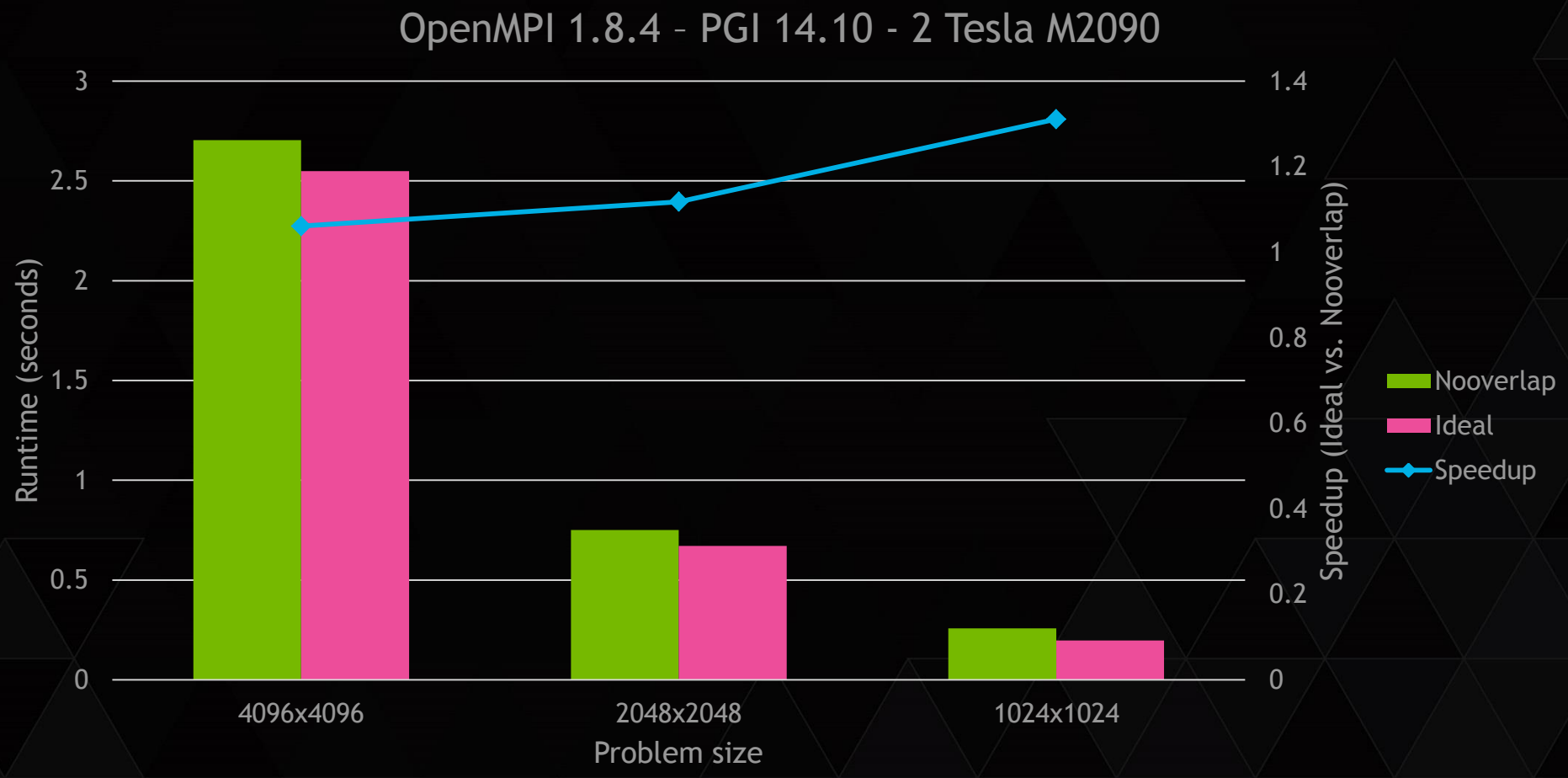


GPU TECHNOLOGY CONFERENCE

PROFILING MPI+OPENACC APPLICATIONS

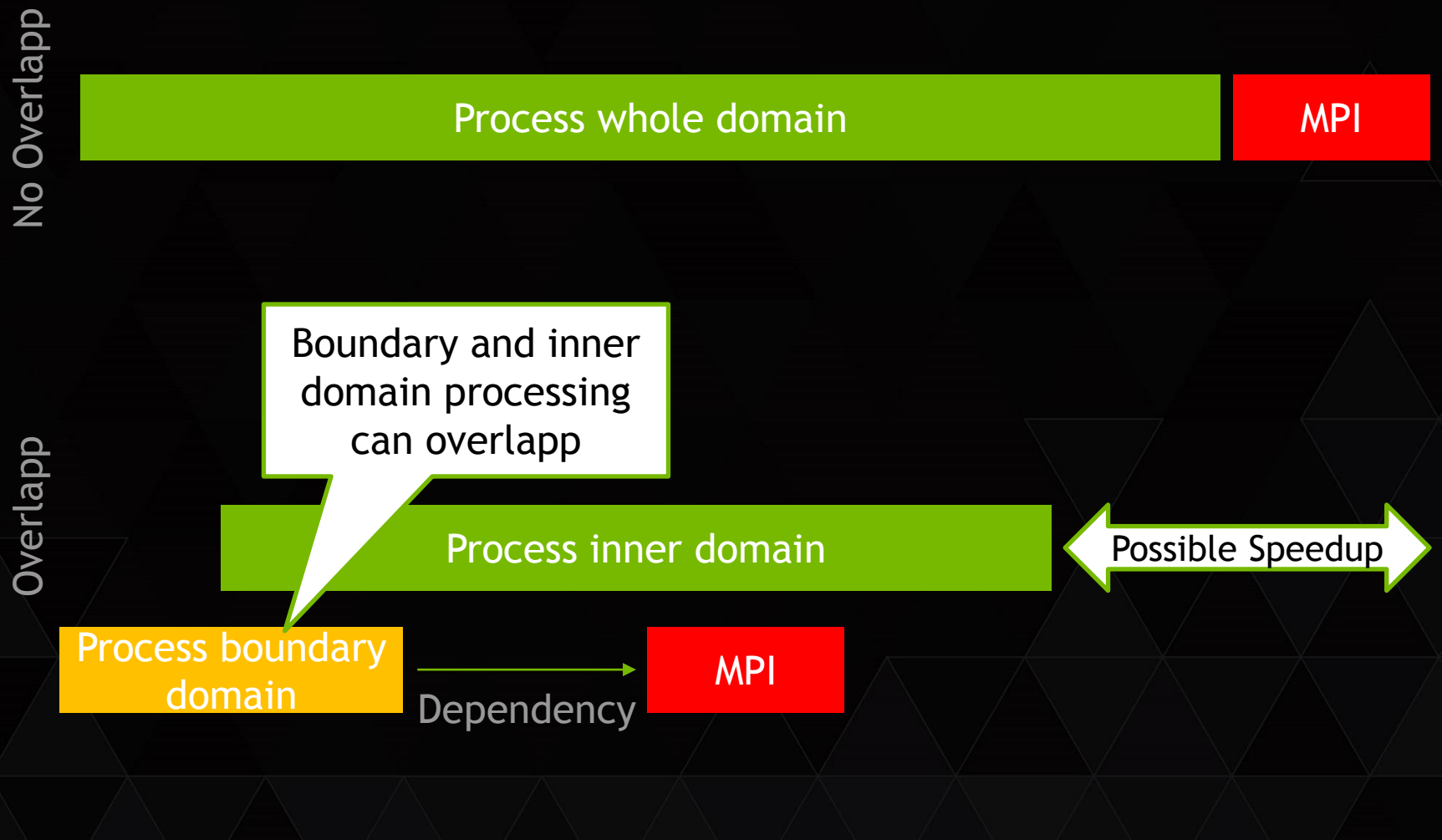


COMMUNICATION + COMPUTATION OVERLAP



GPU TECHNOLOGY CONFERENCE

COMMUNICATION + COMPUTATION OVERLAP



GPU TECHNOLOGY CONFERENCE

COMMUNICATION + COMPUTATION OVERLAP

```
#pragma acc kernels
for ( ... )
      //Process boundary
#pragma acc kernels 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
```

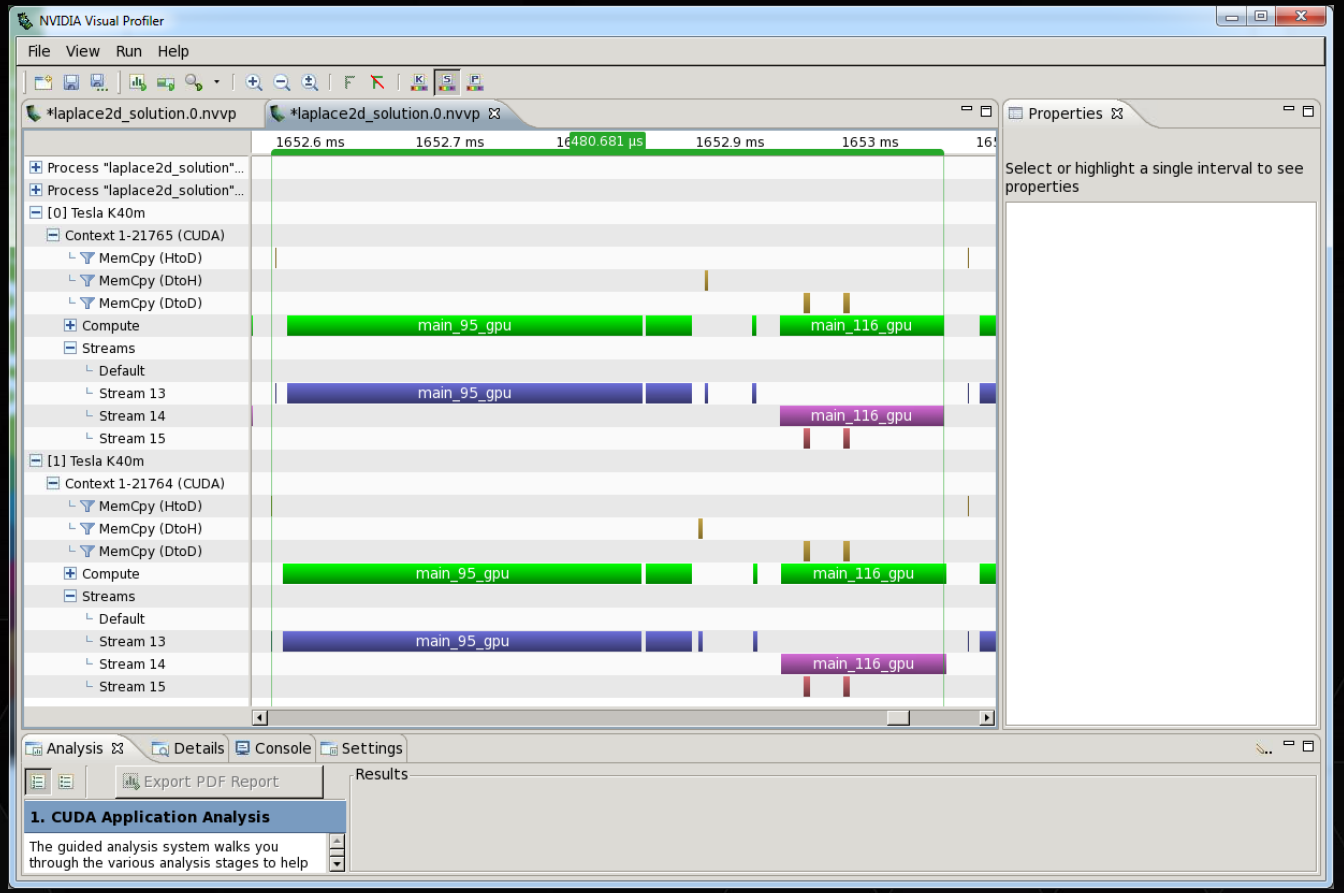
GPU TECHNOLOGY CONFERENCE

TASK3: COMM. + COMPUTATION OVERLAP

- TODOs in `task3/laplace2d.c`
  - Split `Anew` to `A` copy loop in halo and bulk part
  - Launch bulk work asynchronously (`async` clause)
  - Wait for bulk part after MPI (`#pragma acc wait`)

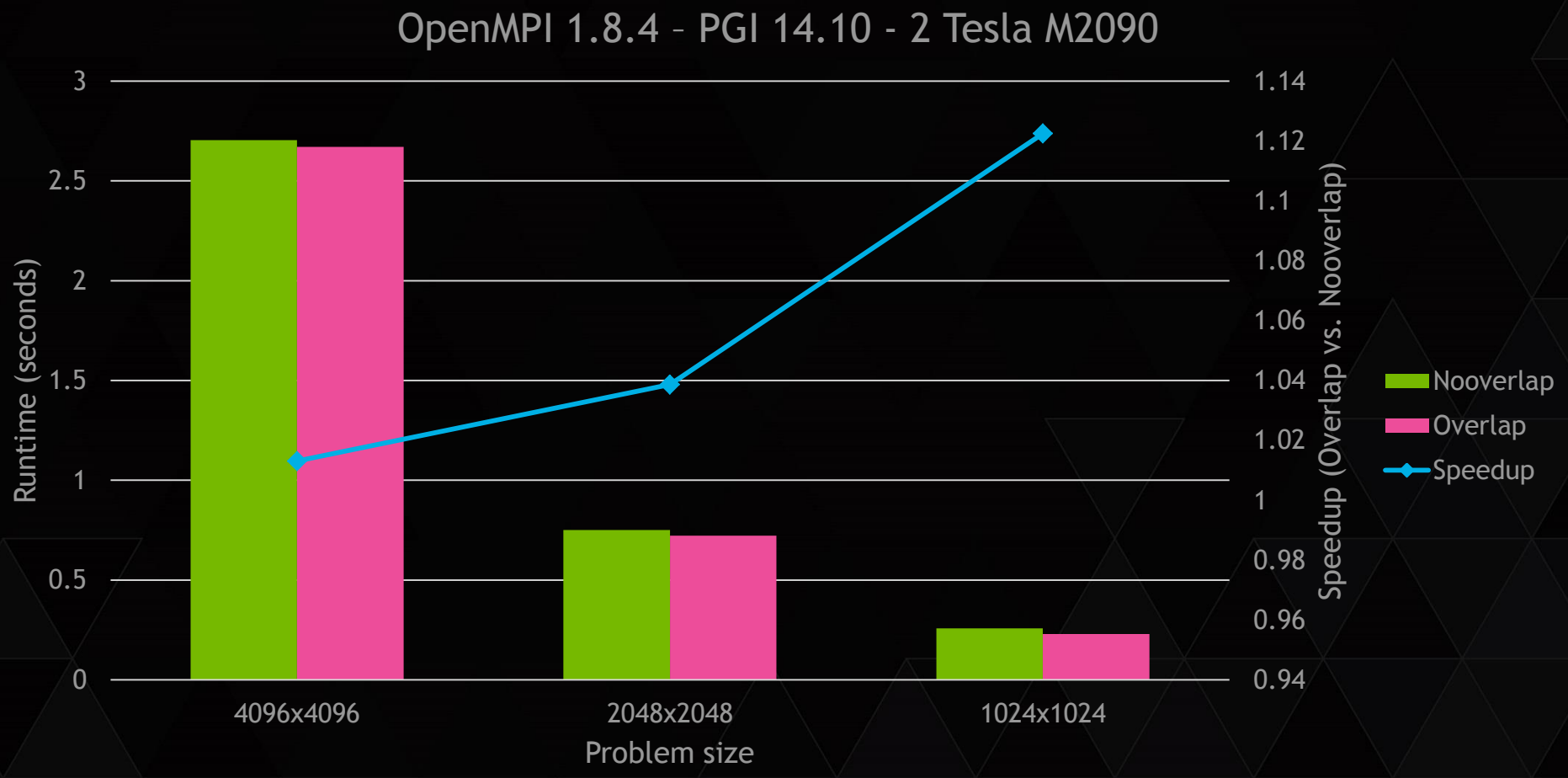
GPU TECHNOLOGY CONFERENCE

PROFILING MPI+OPENACC APPLICATIONS



GPU TECHNOLOGY CONFERENCE

COMMUNICATION + COMPUTATION OVERLAP



GPU TECHNOLOGY CONFERENCE

S5117 - Multi GPU Programming with MPI
(Wednesday 03/18, 15:30 – 16:50, Room 212B)
S5863 - Hangout: OpenACC and Other Directives – (Thursday 03/19, 10:00 - 11:00, Pod C)

THANK YOU

JOIN THE CONVERSATION
#GTC15

https://github.com/jirikraus/Multi_GPU_Programming_with_MPI_and_OpenACC