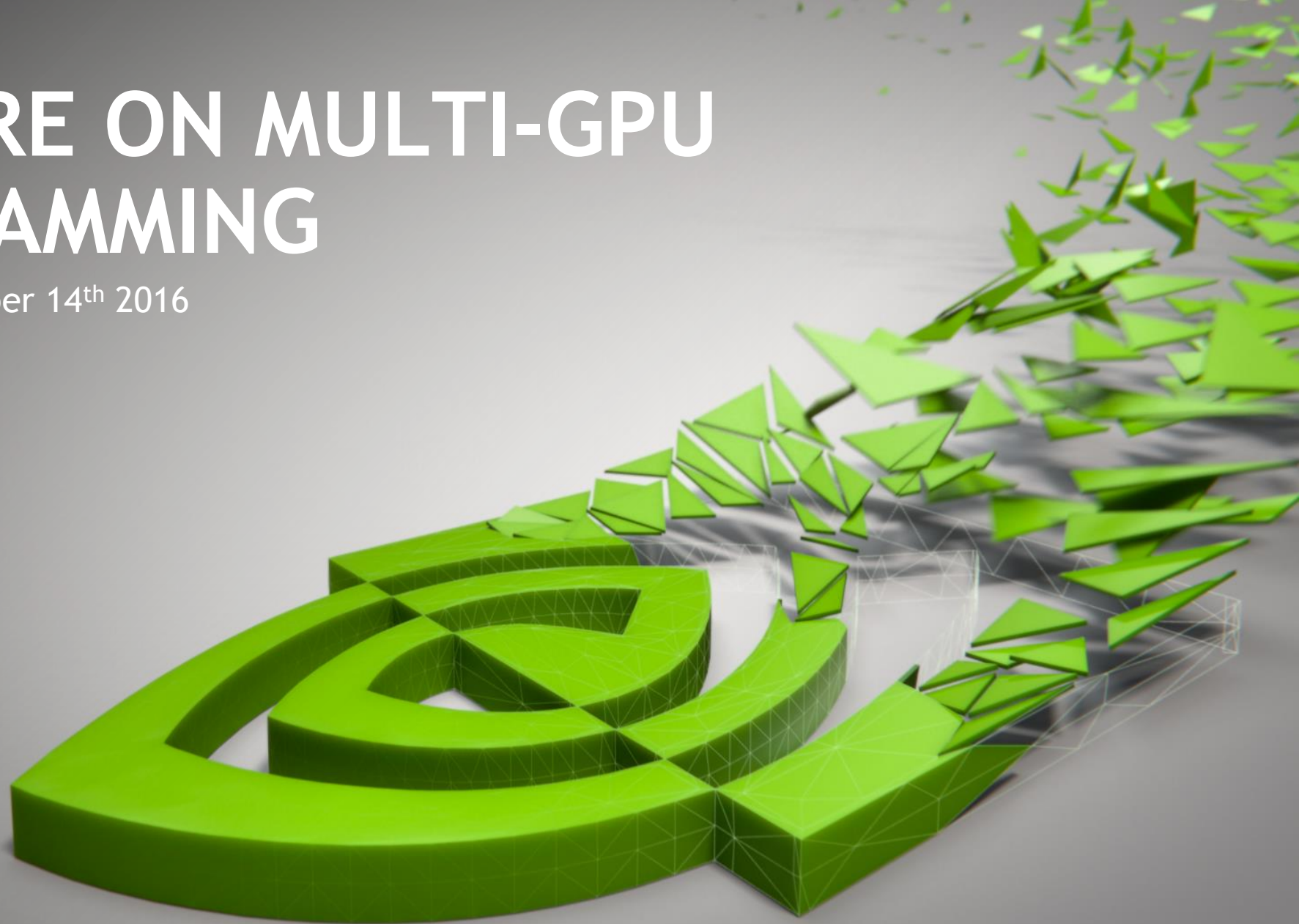


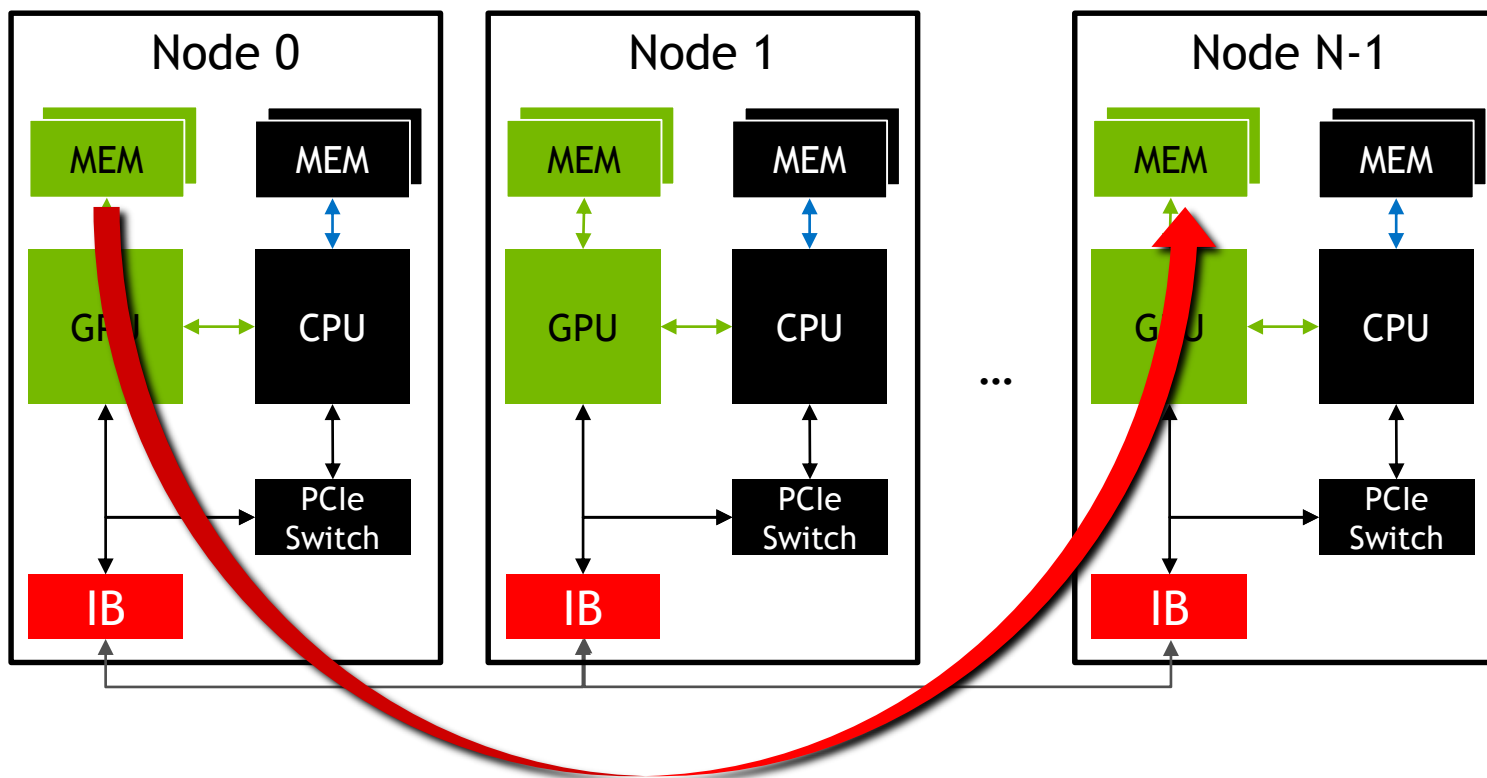
# LECTURE ON MULTI-GPU PROGRAMMING

Jiri Kraus, November 14<sup>th</sup> 2016

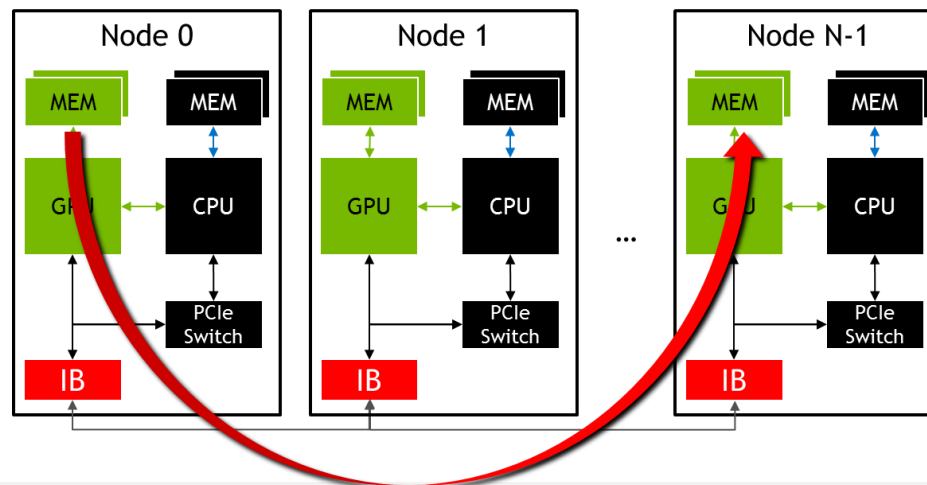


# MULTI GPU PROGRAMMING

With MPI and 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);
```

The background is a solid green color with a subtle, abstract geometric pattern of overlapping triangles and lines, primarily visible on the right side.

Using MPI for inter GPU communication

# 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

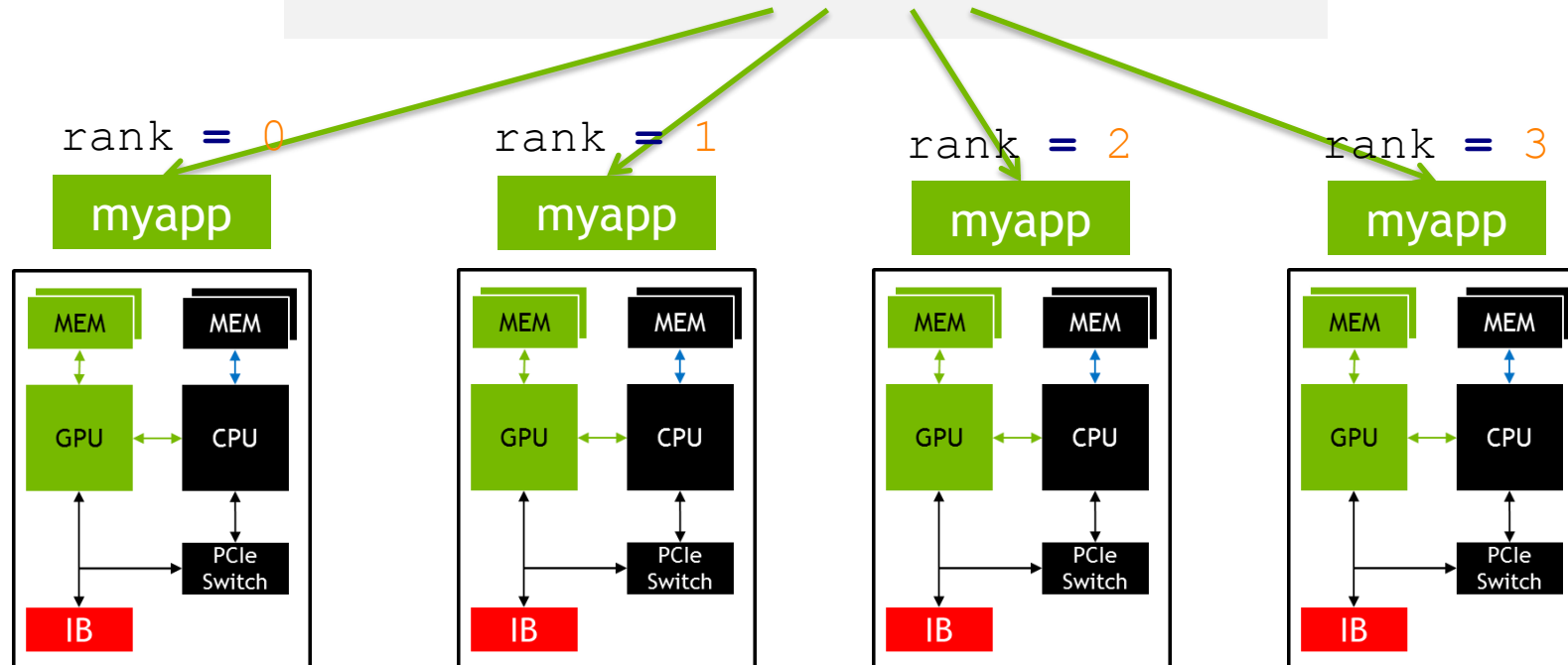
```
#include <mpi.h>

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



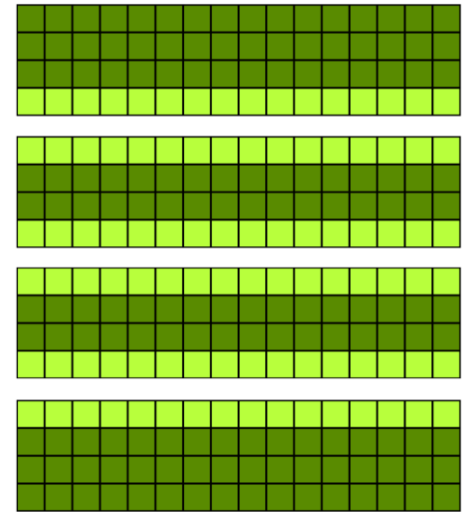
# EXAMPLE: JACOBI SOLVER

Solves the 2D-Poisson Equation on a rectangle

$$\Delta u(x, y) = e^{-10*(x^2+y^2)} \quad \forall (x, y) \in \Omega \setminus \delta\Omega$$

Periodic boundary conditions

Domain decomposition with stripes



Horizontal Stripes



# EXAMPLE: JACOBI SOLVER

## Single GPU

While not converged

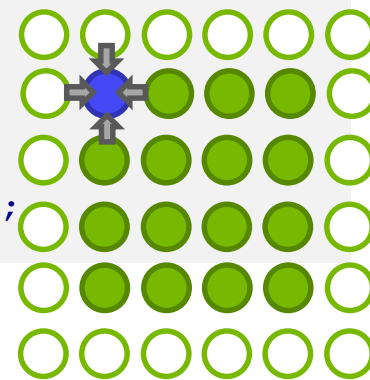
Do Jacobi step:

```
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



# EXAMPLE: JACOBI SOLVER

## Multi GPU

While not converged

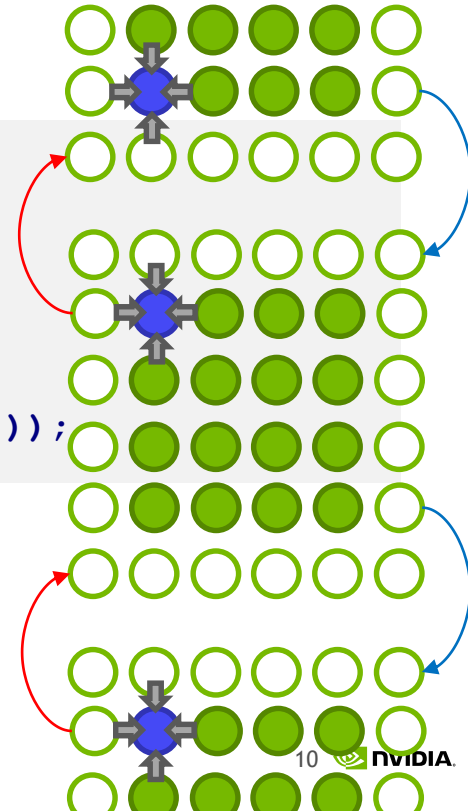
Do Jacobi step:

```
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

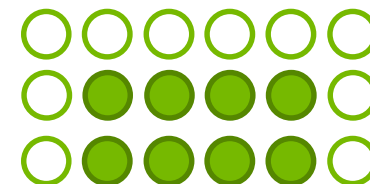
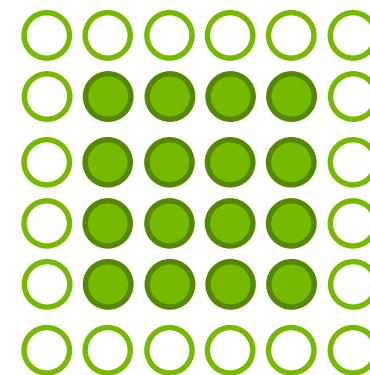
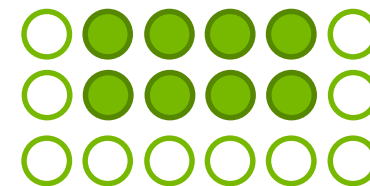


# EXAMPLE JACOBI

## Top/Bottom Halo

```
#pragma acc host_data use_device ( A )  
{
```

```
}
```



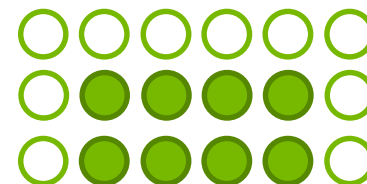
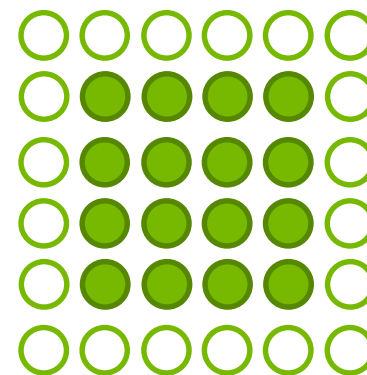
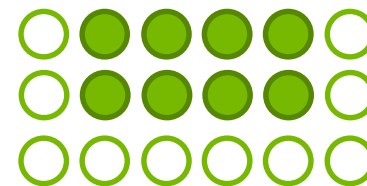
# EXAMPLE JACOBI

## 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);

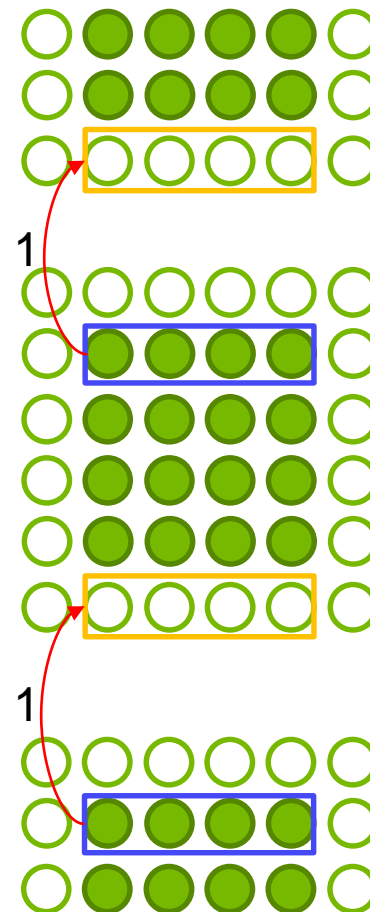
}
```



# 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);  
  
}
```

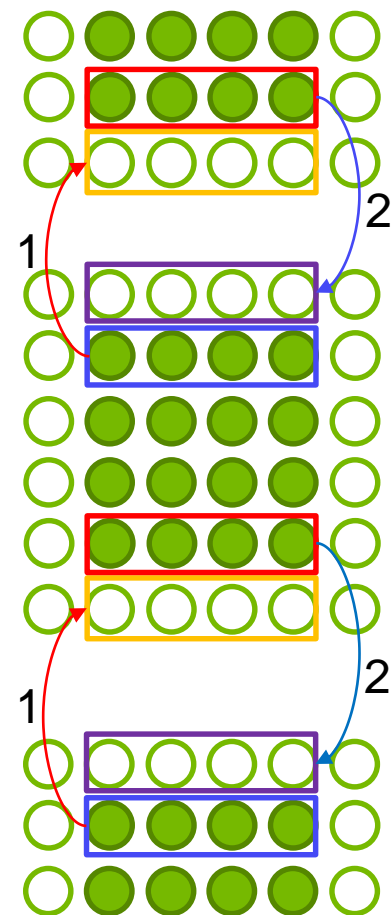


# EXAMPLE JACOBI

## 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 MULTI GPU NODES

## GPU-affinity

```
#pragma acc set device_num( devicenum )

// Or using the API:

#if _OPENACC

acc_device_t device_type = acc_get_device_type();

int ngpus=acc_get_num_devices(device_type);

int devicenum=rank%ngpus;

acc_set_device_num(devicenum,device_type);

#endif /*_OPENACC*/
```

Alternative (OpenMPI):

```
int devicenum = atoi(getenv("OMPI_COMM_WORLD_LOCAL_RANK"));
```

Alternative (MVAPICH2):

```
int devicenum = atoi(getenv("MV2_COMM_WORLD_LOCAL_RANK"));
```

# Profiling of MPI+OpenACC apps



# PROFILING MPI+OPENACC APPLICATIONS

Using pgprof

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

```
mpirun -np $np pgprof --output-profile profile.%q{OMPI_COMM_WORLD_RANK}
```

OpenMPI: OMPI\_COMM\_WORLD\_RANK

MPICH2: MV2\_COMM\_WORLD\_RANK

# PROFILING MPI+OPENACC APPLICATIONS

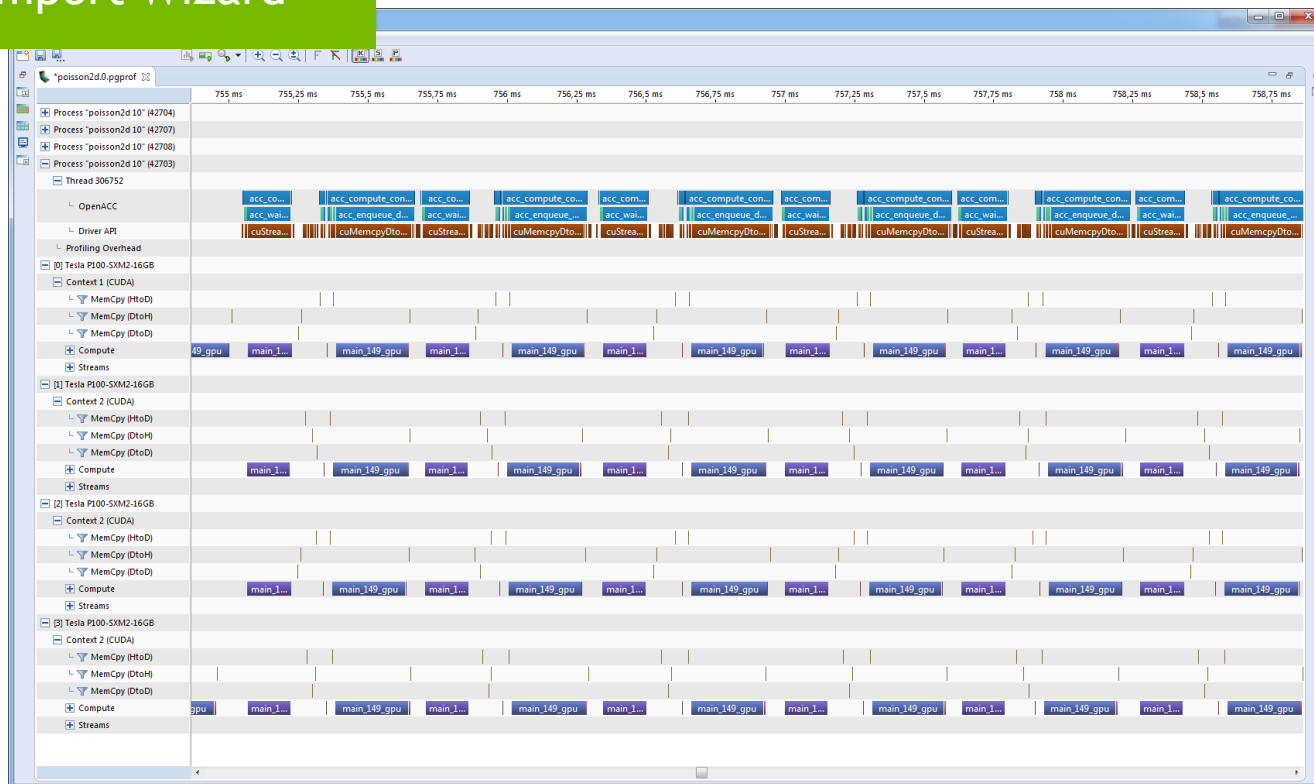
Using pgprof

```
pcp0032@juronb1:~/workspace/sc16-tutorial-openpower/5-Multi_GPU/Tasks/C/task2
[pcp0032@juronb1 task2]$ make profile
bsub -env "all, TMPDIR=/tmp" -n 4 -I -R "rusage[ngpus_share]
ocket 2 -bind-to core -np 4 pgprof --cpu-profiling off -
_WORLD_RANK}.pgprof ./poisson2d 10
Job <4707> is submitted to default queue <interactive>.
<<Waiting for dispatch ...>>
<<Starting on juronc04>>
==42703== PGPROF is profiling process 42703, command: ./
==42704== PGPROF is profiling process 42704, command: ./
==42707== PGPROF is profiling process 42707, command: ./
==42708== PGPROF is profiling process 42708, command: ./
Jacobi relaxation Calculation: 4096 x 4096 mesh
Calculate reference solution and time serial execution.
0, 0.250000
Parallel execution.
0, 0.250000
Num GPUs: 4.
4096x4096: 1 GPU: 0.0305 s, 4 GPUs: 0.0206 s, speedup:
36.98%
MPI time: 0.0037 s, inter GPU BW: 0.33 GiB/s
==42704== Generated result file: /gpfs/homeb/pcp0/pcp0032/workspace/sc16-tutoria
l-openpower/5-Multi_GPU/Tasks/C/task2/poisson2d.0.pgprof
==42708== Generated result file: /gpfs/homeb/pcp0/pcp0032/workspace/sc16-tutoria
l-openpower/5-Multi_GPU/Tasks/C/task2/poisson2d.3.pgprof
==42707== Generated result file: /gpfs/homeb/pcp0/pcp0032/workspace/sc16-tutoria
l-openpower/5-Multi_GPU/Tasks/C/task2/poisson2d.2.pgprof
==42703== Generated result file: /gpfs/homeb/pcp0/pcp0032/workspace/sc16-tutoria
l-openpower/5-Multi_GPU/Tasks/C/task2/poisson2d.1.pgprof
[pcp0032@juronb1 task2]$
```

# PROFILING MPI+OPENACC APPLICATIONS

Using pgprof

Use the import Wizard



# PROFILING MPI+OPENACC APPLICATIONS

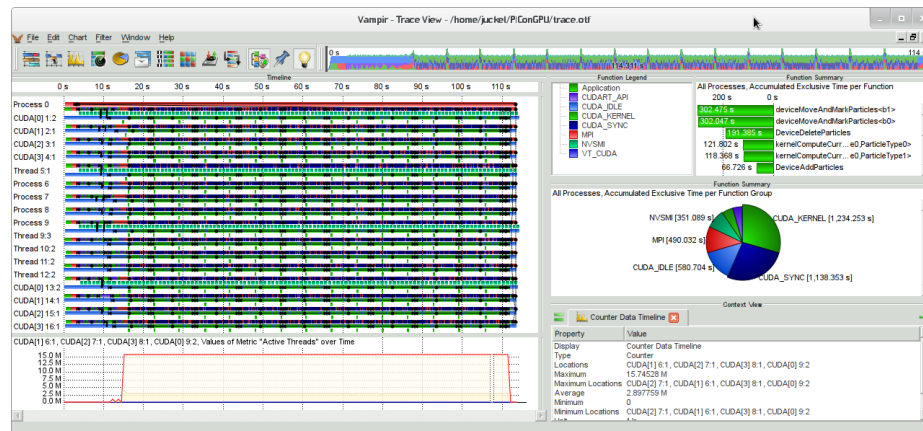
## Third Party Tools

Multiple parallel profiling tools are  
OpenACC-aware

Score-P

Vampir

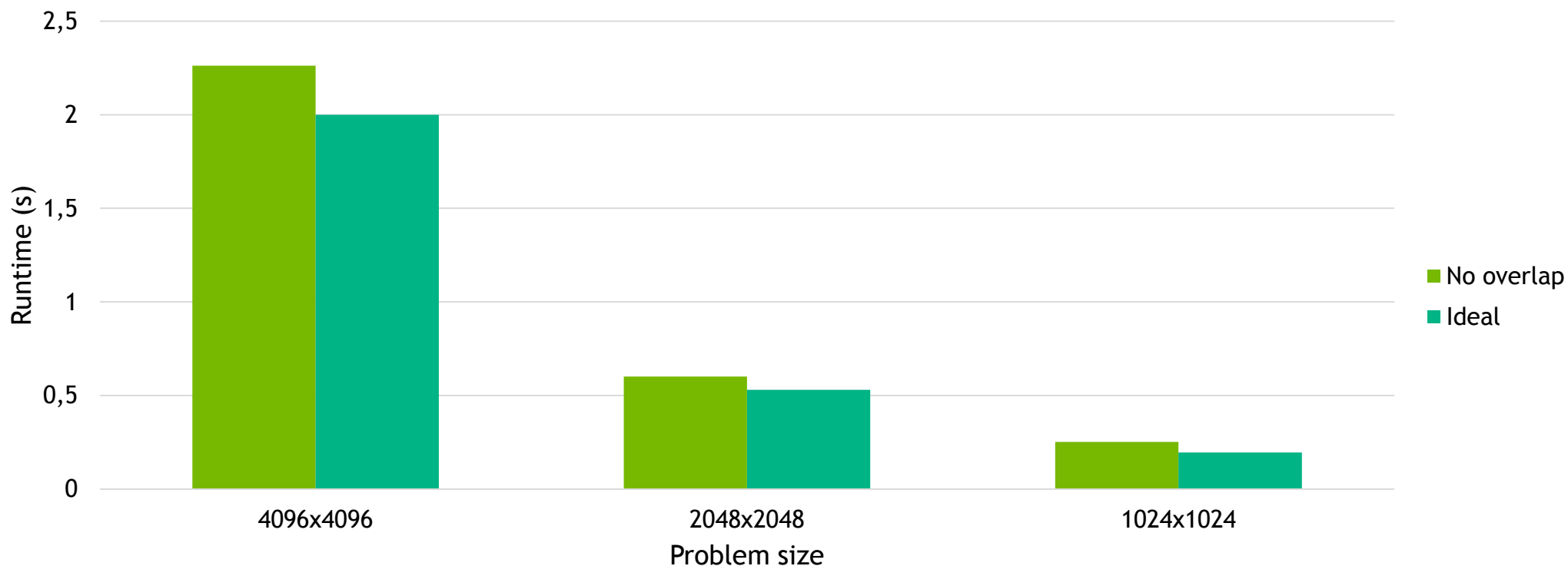
These tools are good for discovering MPI  
issues as well as basic OpenACC performance  
inhibitors.



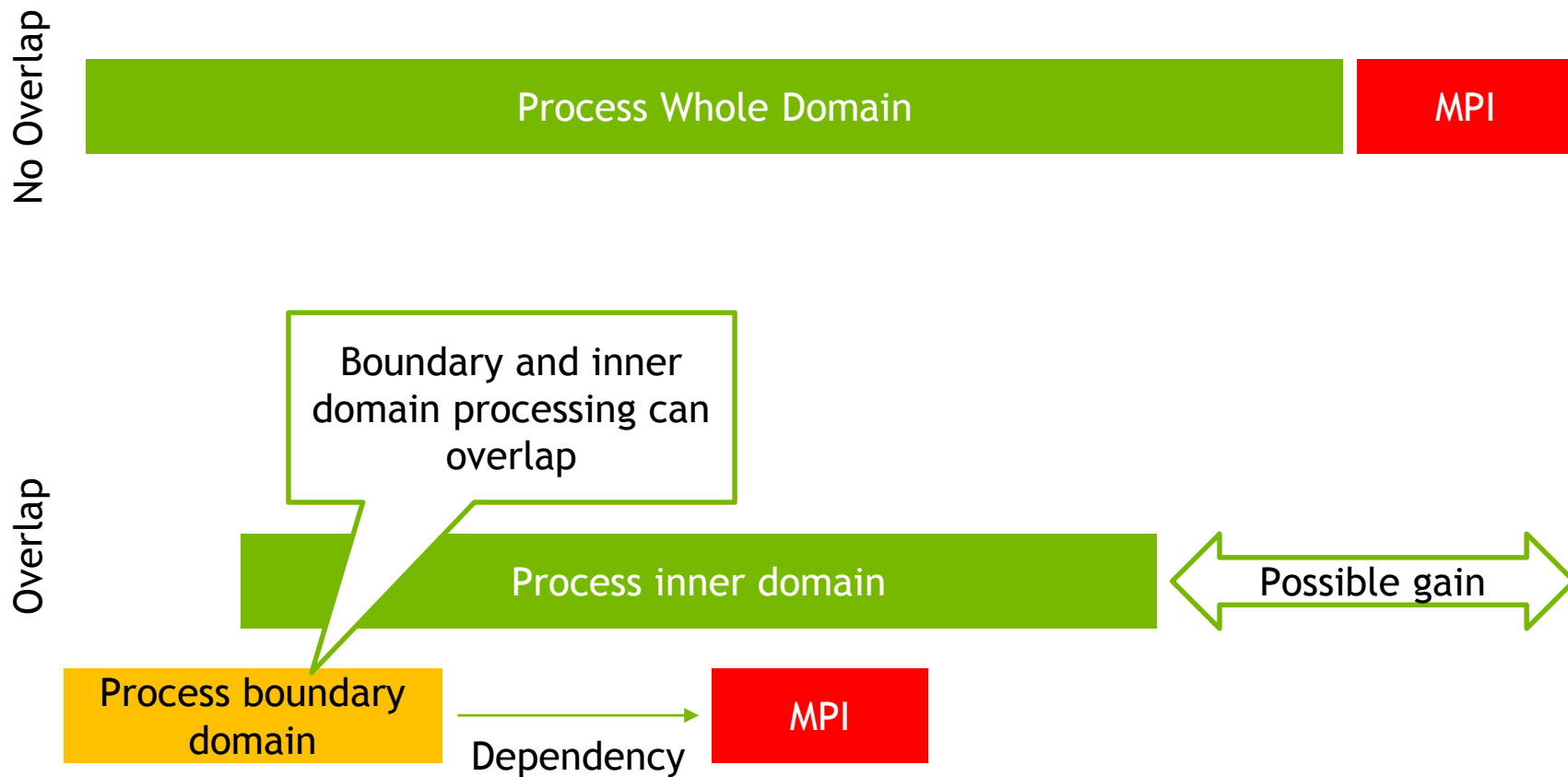
# Overlapping Communication and Computation

# COMMUNICATION + COMPUTATION OVERLAP

OpenMPI 1.10.2 - 2 Tesla K40



# COMMUNICATION + COMPUTATION OVERLAP



# COMMUNICATION + COMPUTATION OVERLAP

## OpenACC with Async Queues

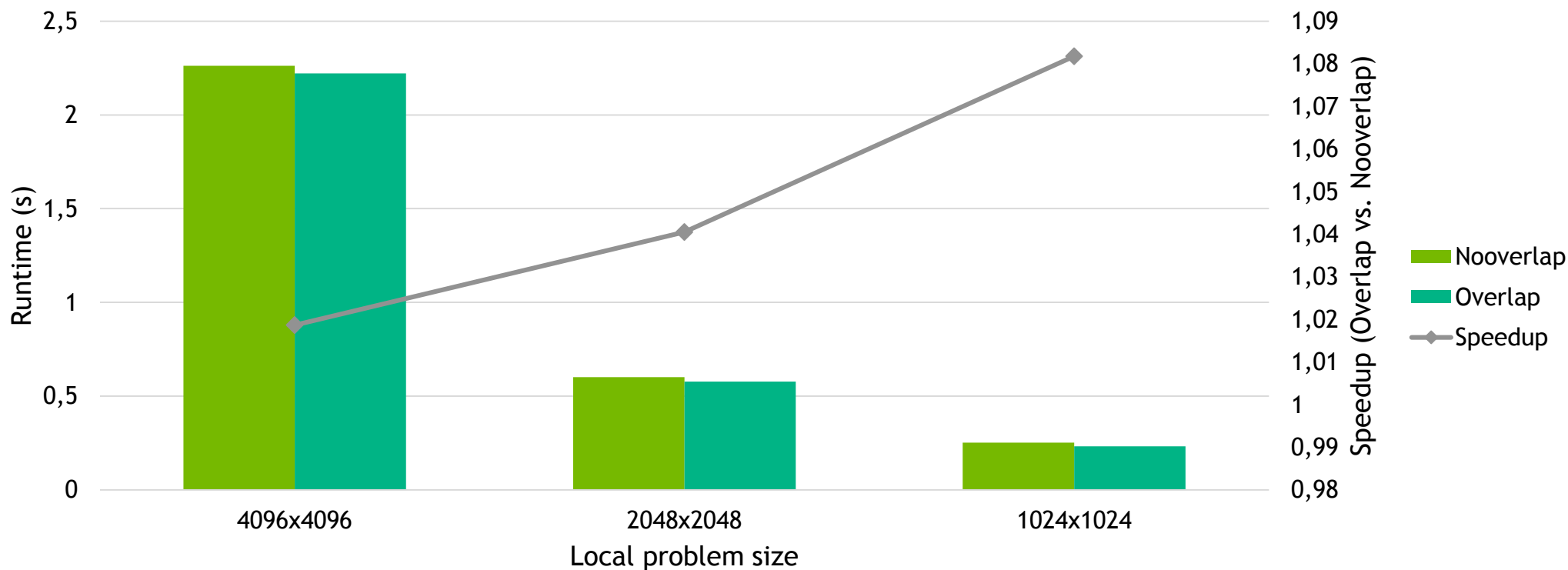
```
#pragma acc parallel loop present(A,Anew)
for ( ... ) //Process boundary

#pragma acc parallel loop present(A,Anew) async
for ( ... ) //Process inner domain
#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);
}
#pragma acc wait //wait for iteration to finish
```



# COMMUNICATION + COMPUTATION OVERLAP

OpenMPI 1.10.2 - 2 Tesla K40



# MPI and Unified Memory

# MPI AND UNIFIED MEMORY

## CAVEAT

Using Unified Memory with a non Unified Memory-aware MPI might break in some cases, e.g. when registering memory for RDMA, or even worse silently produce wrong results.



**Use a Unified Memory-aware MPI with Unified Memory and MPI**

Unified Memory-aware: CUDA-aware MPI with support for Unified Memory

# MPI AND UNIFIED MEMORY

## Current Status

Available Unified Memory-aware MPI implementations

- OpenMPI (since 1.8.5)
- MVAPICH2-GDR (since 2.2b)
  - Performance improvements with 2.2RC1 for Intranode GPU to GPU communication

Currently both treat all Unified Memory as Device Memory



Good performance if all buffers used in MPI are touched mainly on the GPU.

# MPI AND UNIFIED MEMORY

## Without Unified Memory-aware MPI

Only use non Unified Memory Buffers for MPI: cudaMalloc, cudaMallocHost or malloc

Application managed non Unified Memory Buffers also allow to work around current missing cases in Unified Memory-aware MPI Implementations.