

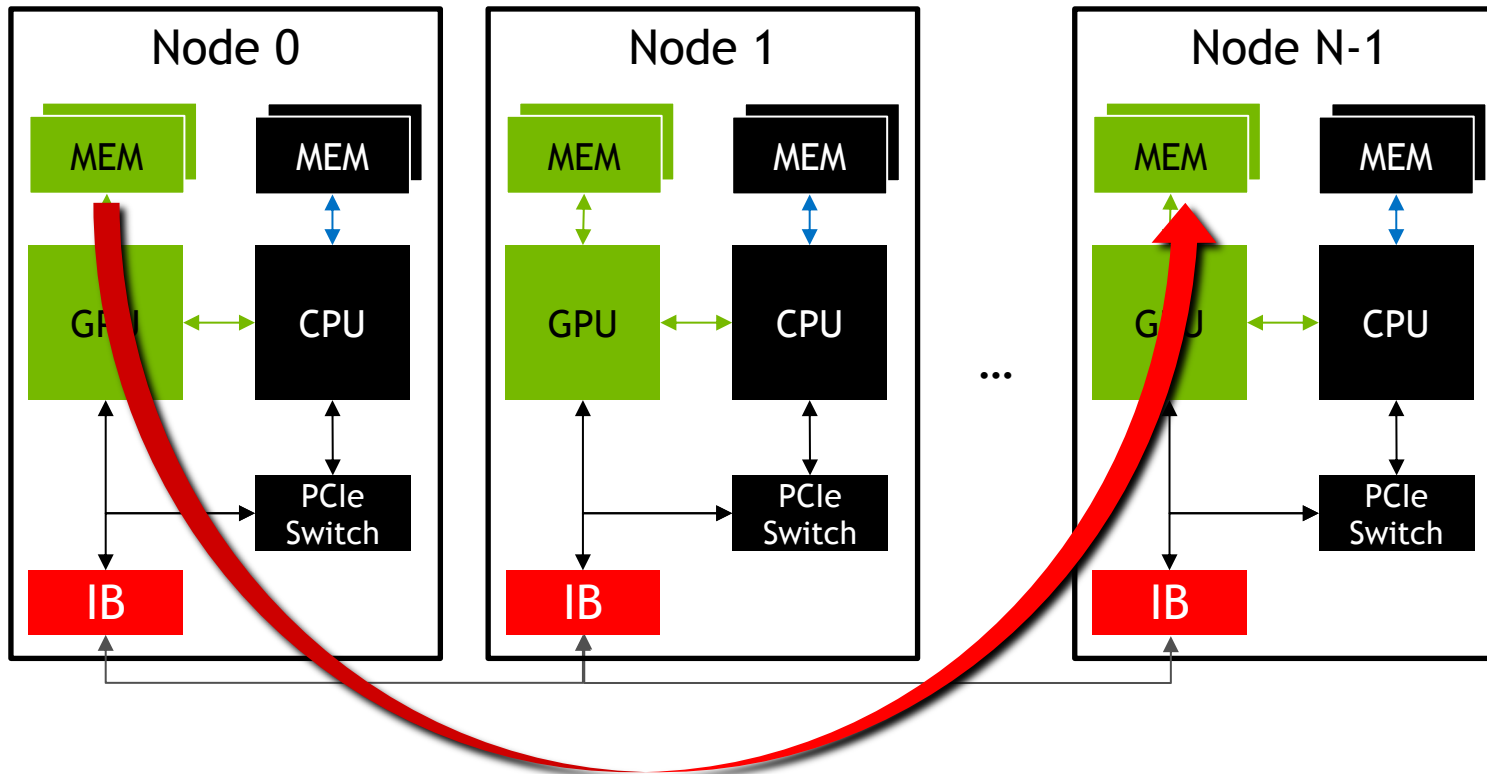


LECTURE ON MULTI-GPU PROGRAMMING

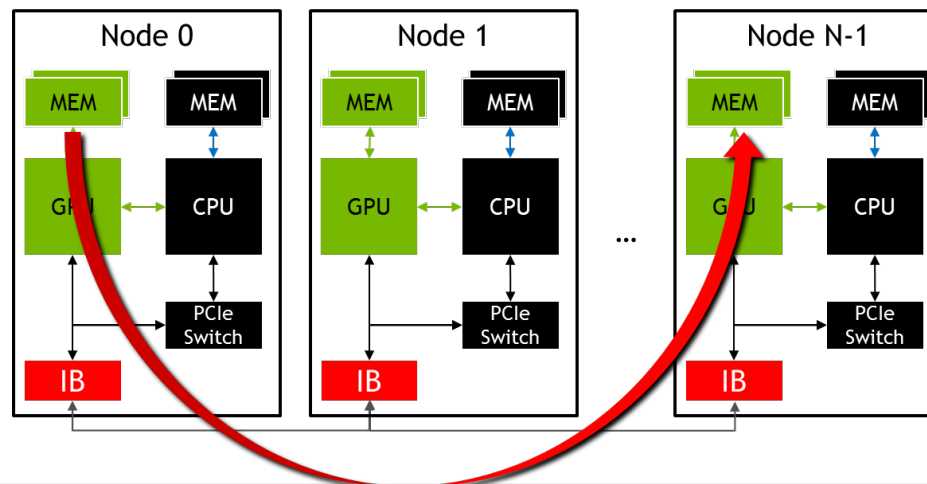
Mathias Wagner, November 18th 2019

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 of the slide features an abstract network diagram. It consists of numerous small, glowing green circular nodes scattered across a dark, almost black, background. These nodes are interconnected by a dense web of thin, light green lines, creating a complex, web-like structure that suggests a network or data flow. The lines vary in length and orientation, some connecting nearby nodes while others span larger distances. The overall effect is one of a dynamic, interconnected system.

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 Spectrum MPI, Cray MPT, ...

MPI - SKELETON

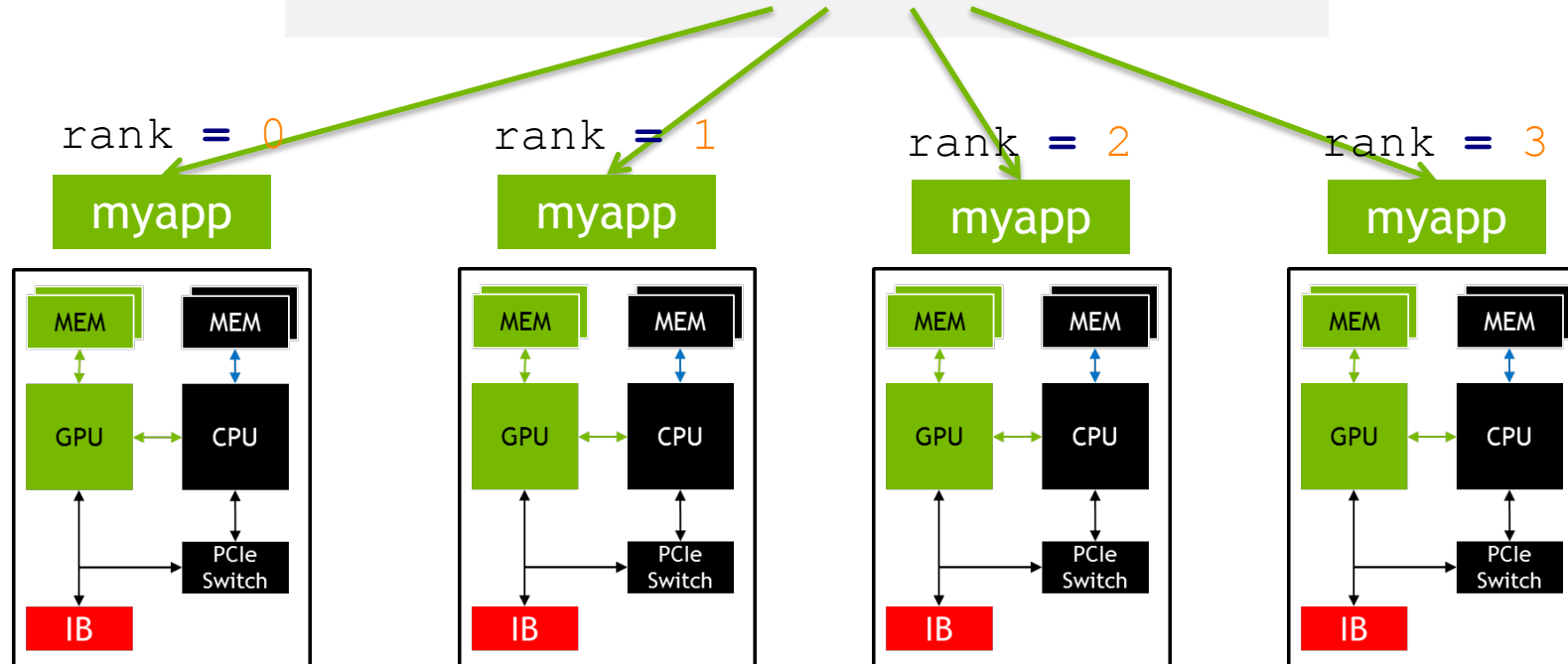
```
#include <mpi.h>

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>
```



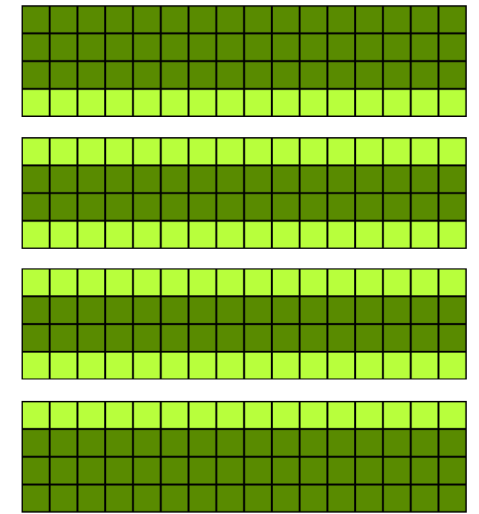
EXAMPLE: JACOBI SOLVER

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

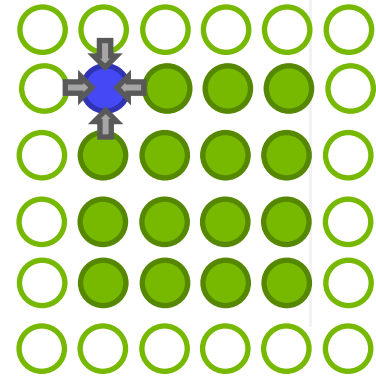
Do Jacobi step:

```
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

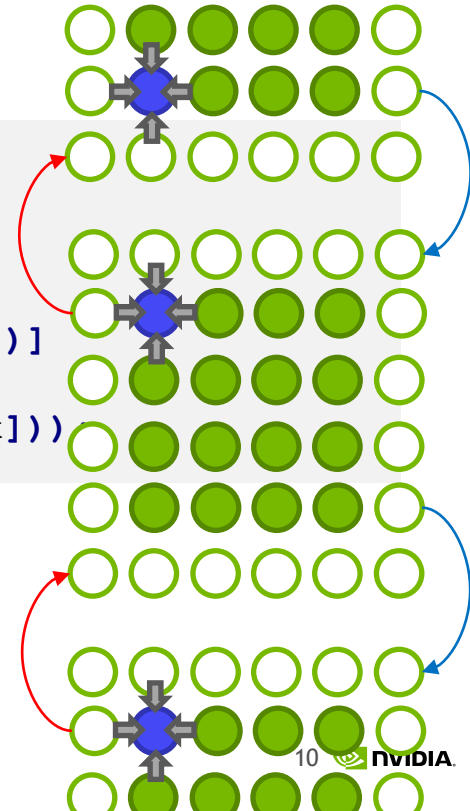
Do Jacobi step:

```
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

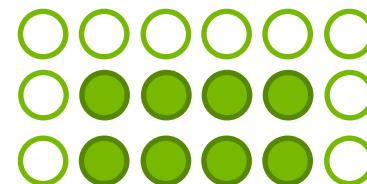
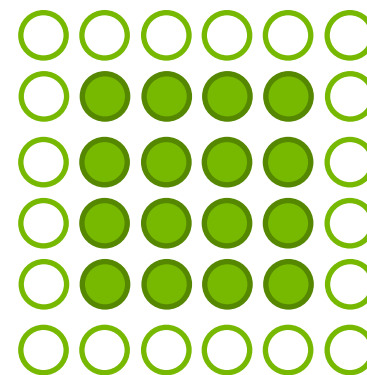
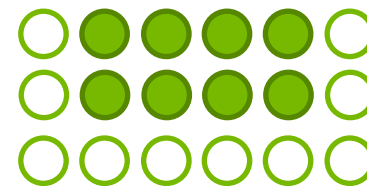


EXAMPLE JACOBI

Top/Bottom Halo

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

```
}
```



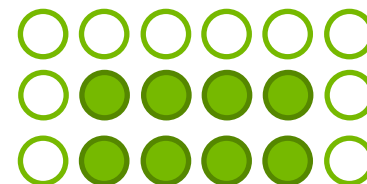
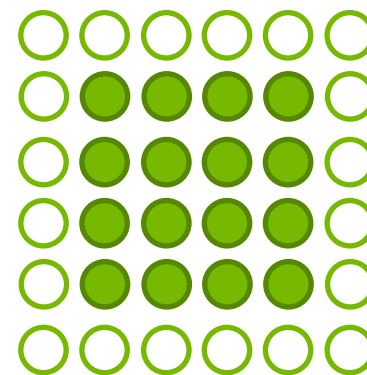
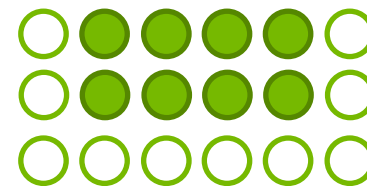
EXAMPLE JACOBI

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);

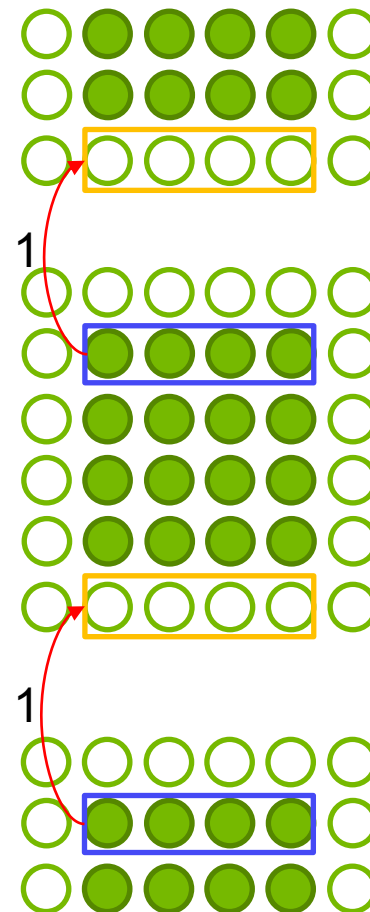
}
```



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);  
  
}
```

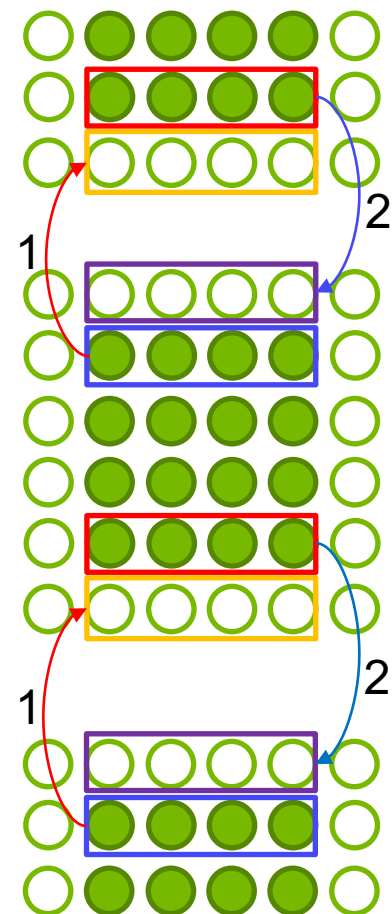


EXAMPLE JACOBI

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 / Spectrum MPI):

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

Alternative (MVAPICH2):

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

The background of the slide features an abstract network diagram. It consists of numerous small, bright green circular nodes scattered across a dark blue to black gradient. These nodes are interconnected by a dense web of thin, light green lines, creating a complex, web-like structure that suggests a network or data flow. The lines vary in opacity and thickness, adding depth to the visual.

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}.nvvp
```

OpenMPI / Spectrum MPI: OMPI_COMM_WORLD_RANK

MVAPICH2: MV2_COMM_WORLD_RANK

Additionally highlight MPI activity in the timeline

```
mpirun -np $np pgprof --annotate-mpi openmpi --output-profile ...
```

OpenMPI / Spectrum MPI : --annotate-mpi openmpi

MVAPICH2: --annotate-mpi mpich

PROFILING MPI+OPENACC APPLICATIONS

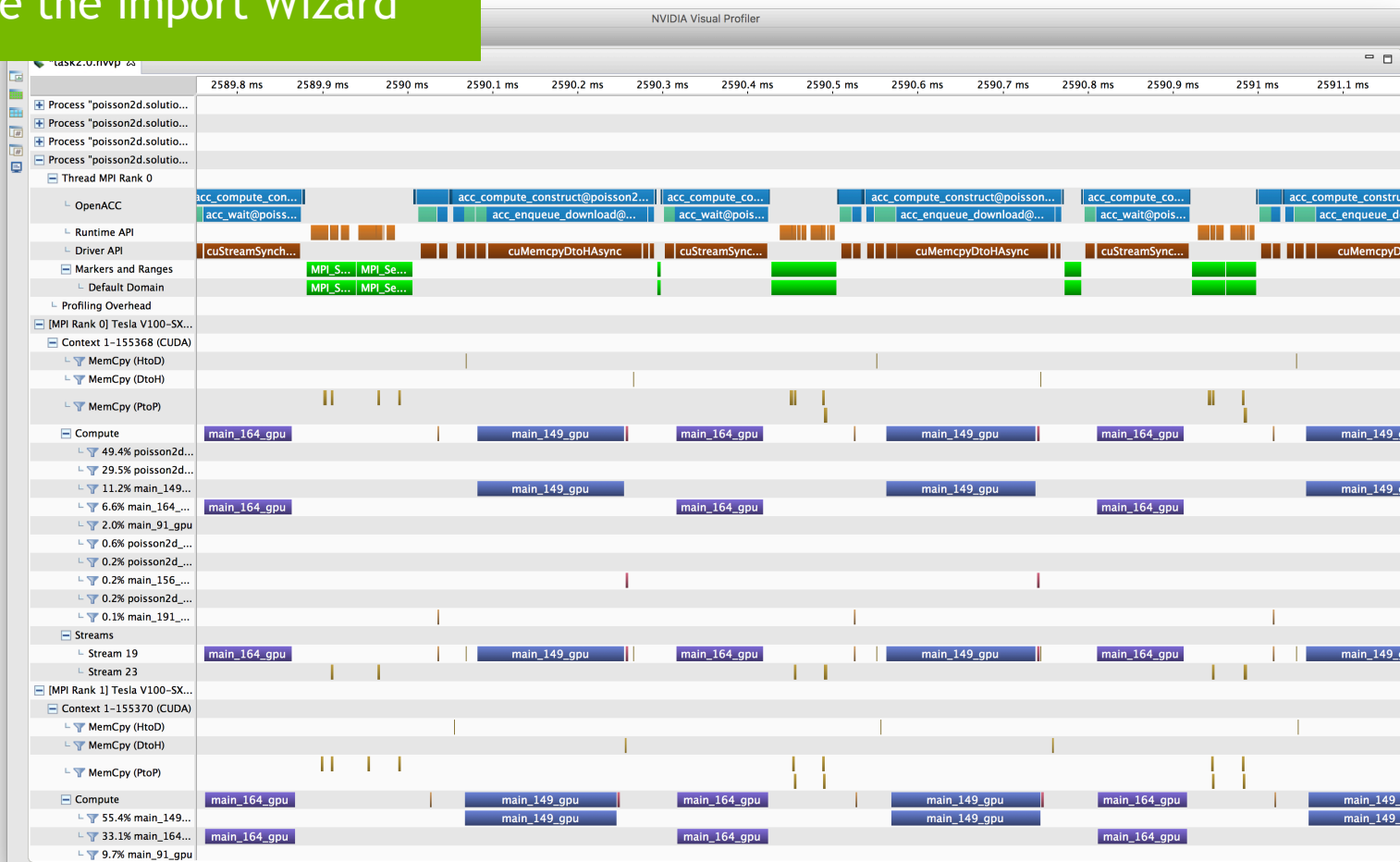
Using pgprof

```
ssh %1
bash-4.2$ jsrun --smpiargs "-gpu" -n1 -g ALL_GPUS -c ALL_CPUS -a4 pgprof --annotate-mpi openmpi --cpu-pro
filing off -o /gpfs/wolf/gen110/proj-shared/mathiasw/task2.%q{OMPI_COMM_WORLD_RANK}.nvvp ./poisson2d.solu
tion
==155052== PGPROF is profiling process 155052, command: ./poisson2d.solution
==155054== PGPROF is profiling process 155054, command: ./poisson2d.solution
==155055== PGPROF is profiling process 155055, command: ./poisson2d.solution
==155053== PGPROF is profiling process 155053, command: ./poisson2d.solution
==155052== Generated result file: /gpfs/wolf/gen110/proj-shared/mathiasw/task2.3.nvvp
==155054== Generated result file: /gpfs/wolf/gen110/proj-shared/mathiasw/task2.2.nvvp
==155053== Generated result file: /gpfs/wolf/gen110/proj-shared/mathiasw/task2.1.nvvp
==155055== Generated result file: /gpfs/wolf/gen110/proj-shared/mathiasw/task2.0.nvvp
Jacobi relaxation Calculation: 4096 x 4096 mesh
Calculate reference solution and time serial execution.
 0, 0.250000
100, 0.249940
200, 0.249880
300, 0.249821
400, 0.249761
500, 0.249702
600, 0.249642
700, 0.249583
800, 0.249524
900, 0.249464
Parallel execution.
 0, 0.250000
100, 0.249940
200, 0.249880
300, 0.249821
400, 0.249761
500, 0.249702
600, 0.249642
700, 0.249583
800, 0.249524
900, 0.249464
Num GPUs: 4.
4096x4096: 1 GPU: 1.3687 s, 4 GPUs: 0.5355 s, speedup: 2.56, efficiency: 63.90%
MPI time: 0.0882 s, inter GPU BW: 1.38 GiB/s
bash-4.2$
```

PROFILING MPI+OPENACC APPLICATIONS

Using pgprof

Use the import Wizard



PROFILING MPI+OPENACC APPLICATIONS

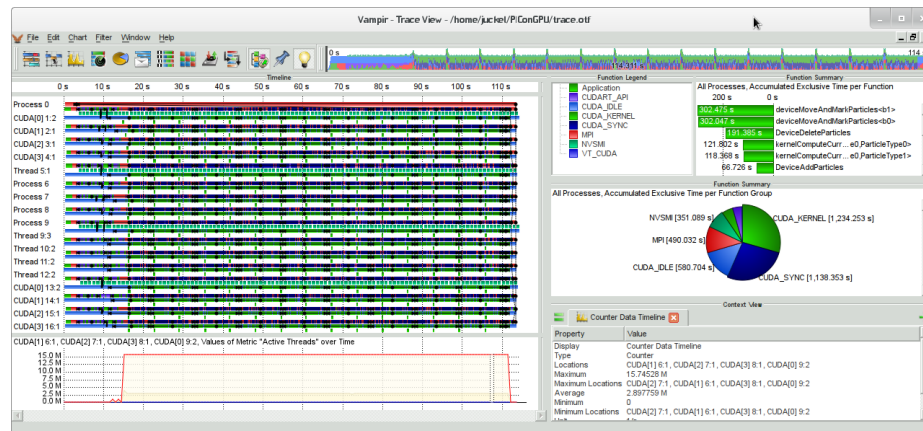
Third Party Tools


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.

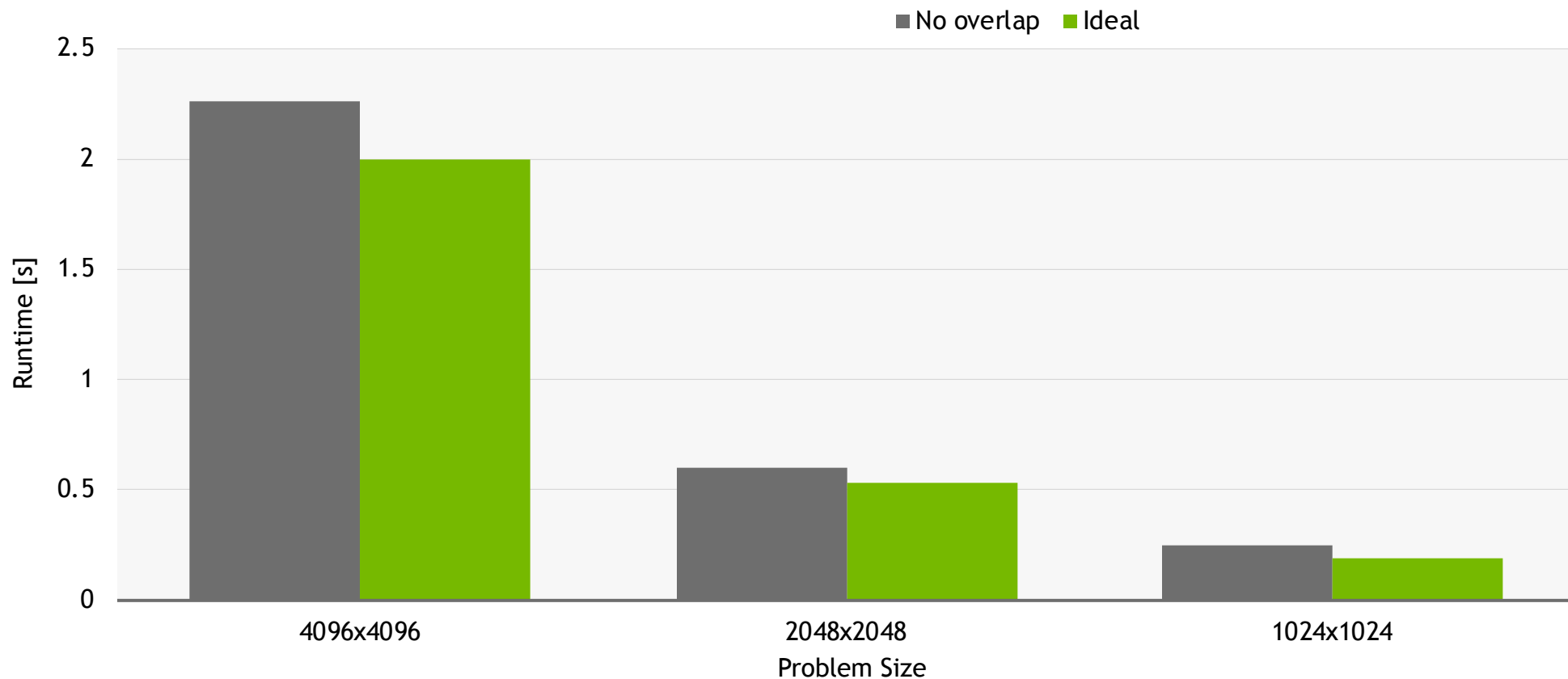


An abstract network diagram with a dark blue background. It features numerous thin, light green lines connecting various nodes. The nodes are represented by small, bright green circles of varying sizes. Some nodes are larger and more prominent, while others are smaller and less distinct. The lines crisscross the frame, creating a complex web of connections. The overall effect is one of a dynamic, interconnected system.

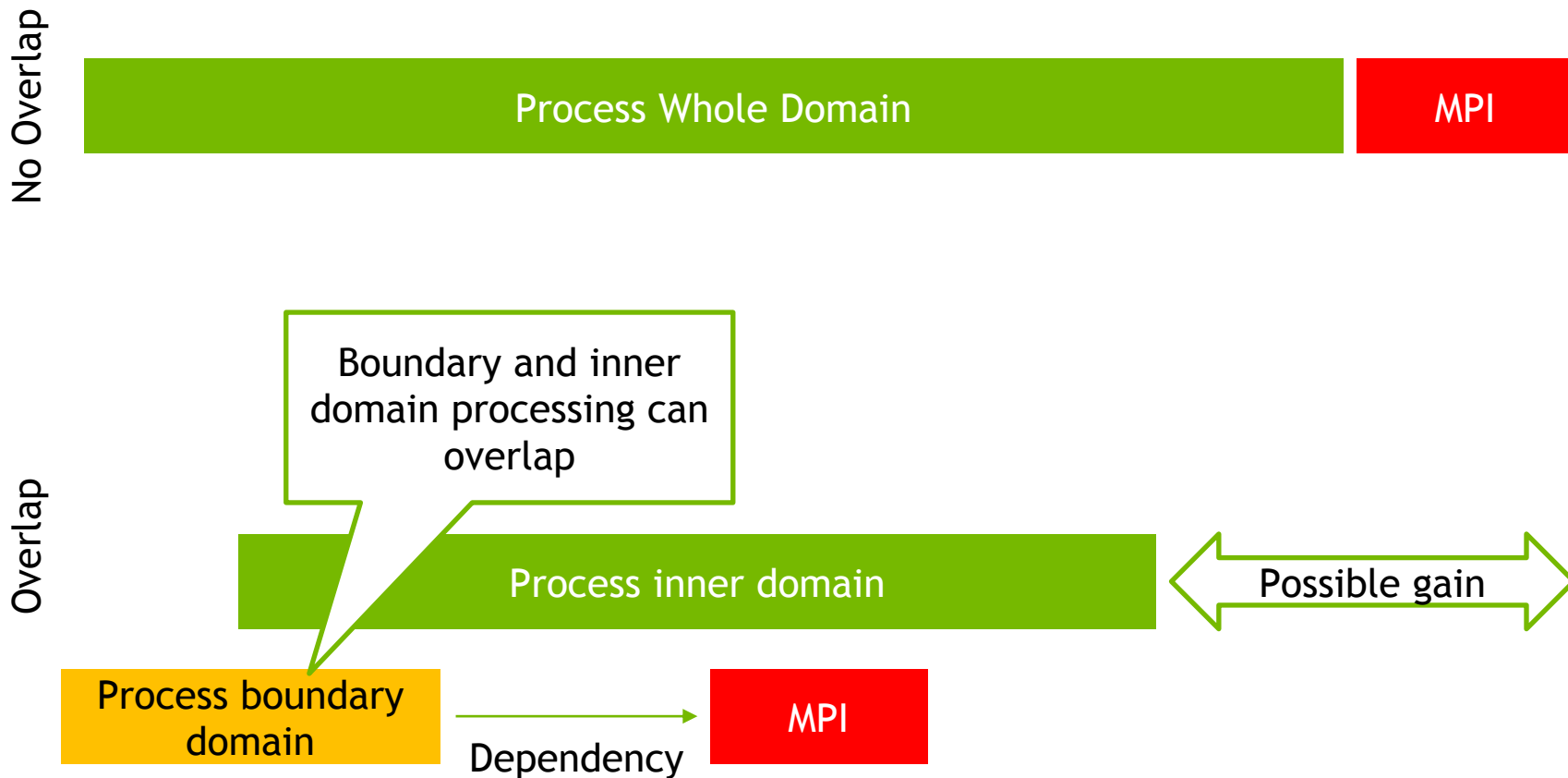
OVERLAPPING COMMUNICATION AND COMPUTATION

COMMUNICATION + COMPUTATION OVERLAP

OpenMPI 2.1.2 - 2 Quadro GV100



COMMUNICATION + COMPUTATION OVERLAP



COMMUNICATION + COMPUTATION OVERLAP

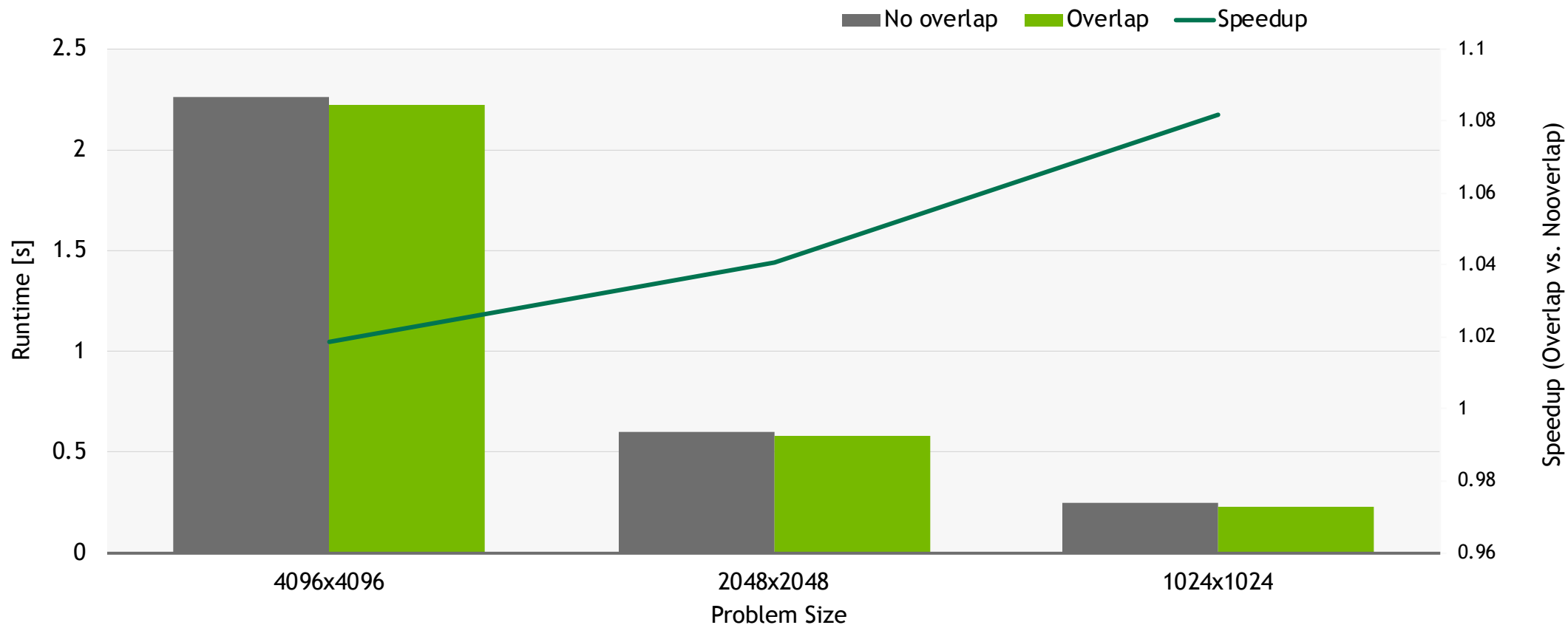
OpenACC with Async Queues

```
#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 2.1.2 - 2 Quadro GV100



An abstract network diagram with green nodes and lines on a dark background. The nodes are represented by small green circles of varying sizes, some of which are glowing. They are interconnected by a dense web of thin, light green lines. The background is dark blue/black with some larger, faint green circular shapes.

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)
- Spectrum MPI

Currently 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.

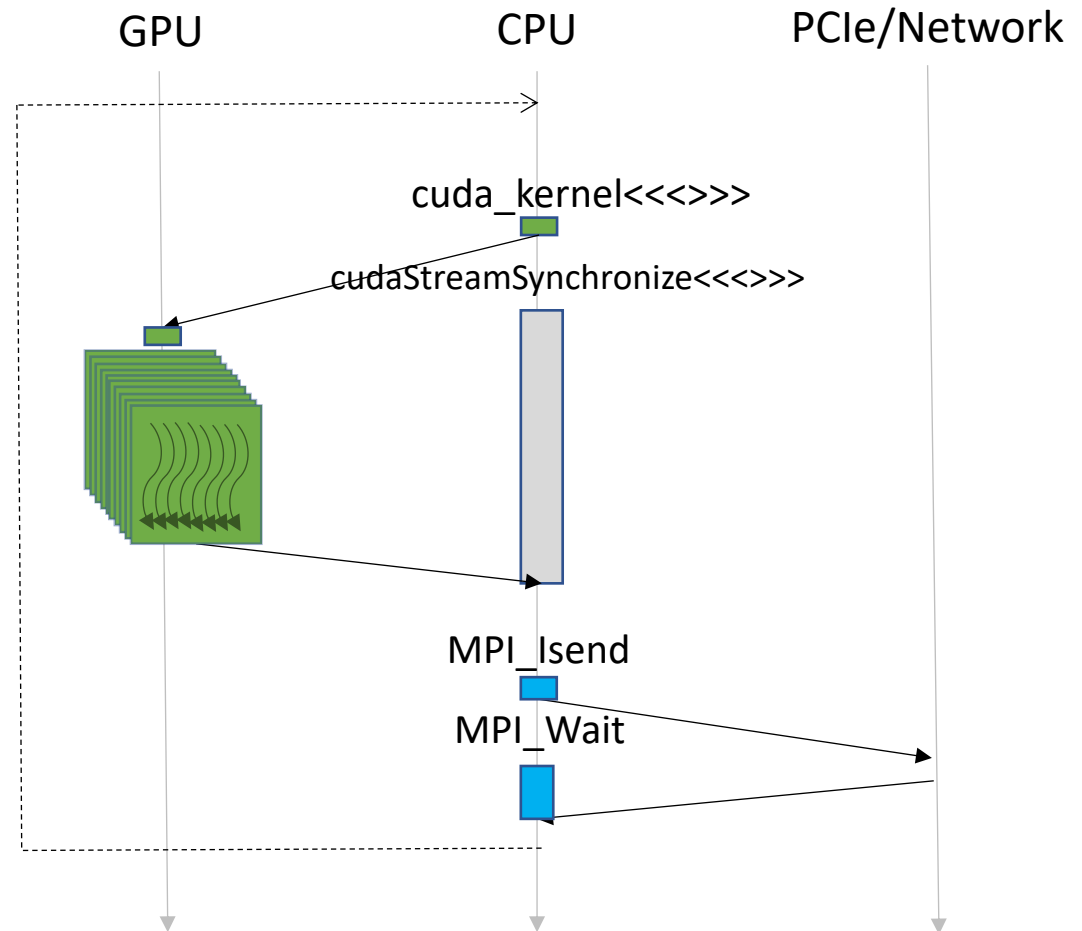
The background is a dark blue field with a complex network of thin, light green lines crisscrossing across it. At various points where these lines intersect or terminate, there are small, bright green circular dots. Some of these dots have a soft, out-of-focus glow around them, while others are sharper. The overall effect is one of a dynamic, interconnected system, possibly representing a network or data flow.

**LOOKING FORWARD
NVSHMEM**

GPU FOR COMPUTE OFFLOAD

Compute on GPU
Communication from CPU
Synchronization at boundaries

Performance overheads
Offload latencies
Synchronization overheads



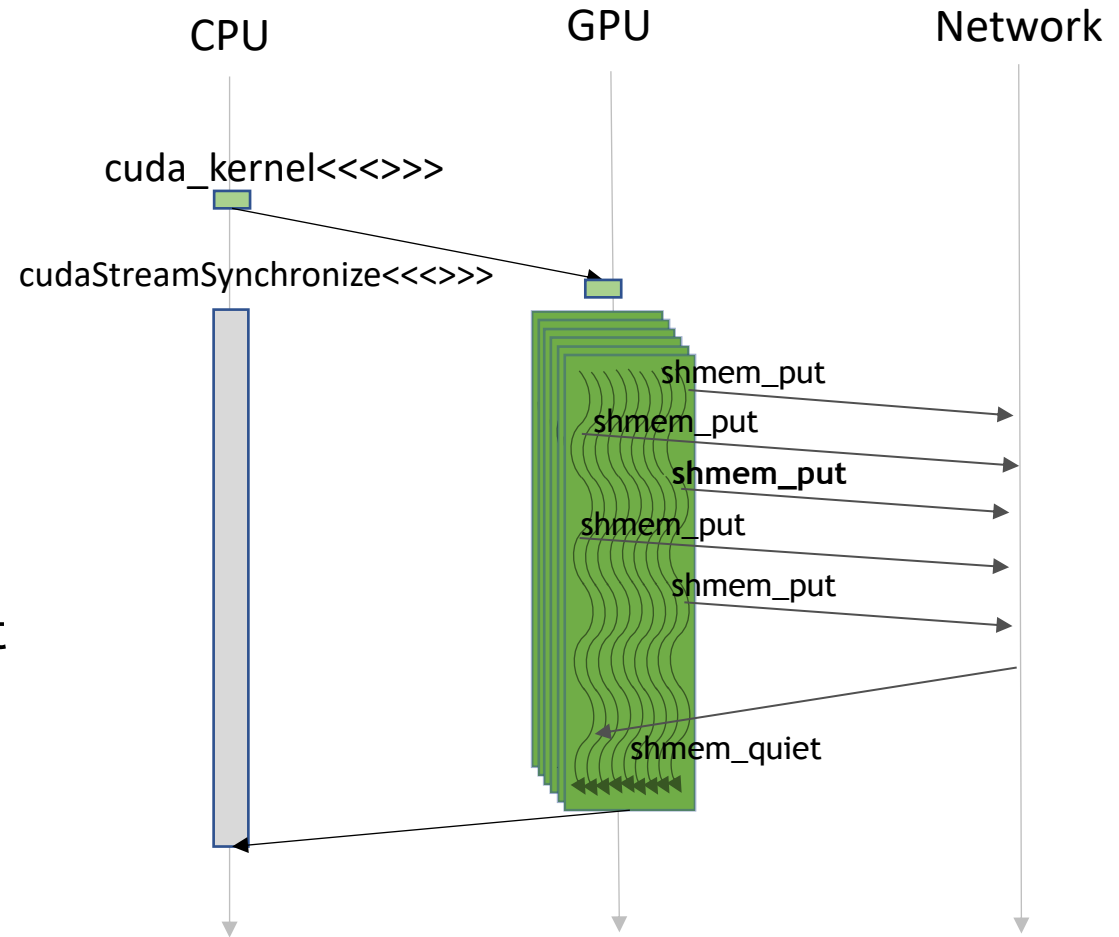
GPU MASTERS COMMUNICATION

Compute - communication overlap

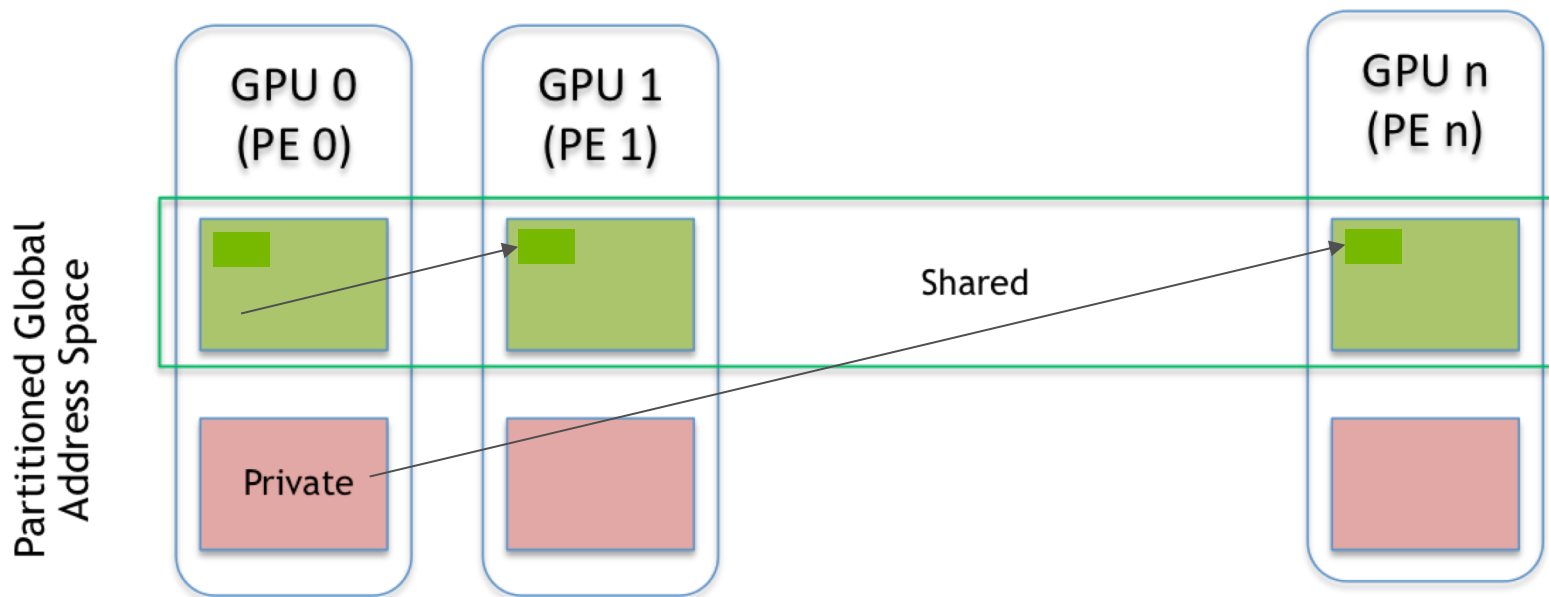
Easier to write algorithms with inline communication

Reduce reliance on CPU

Improving performance while making it easier to program



WHAT IS NVSHMEM ?



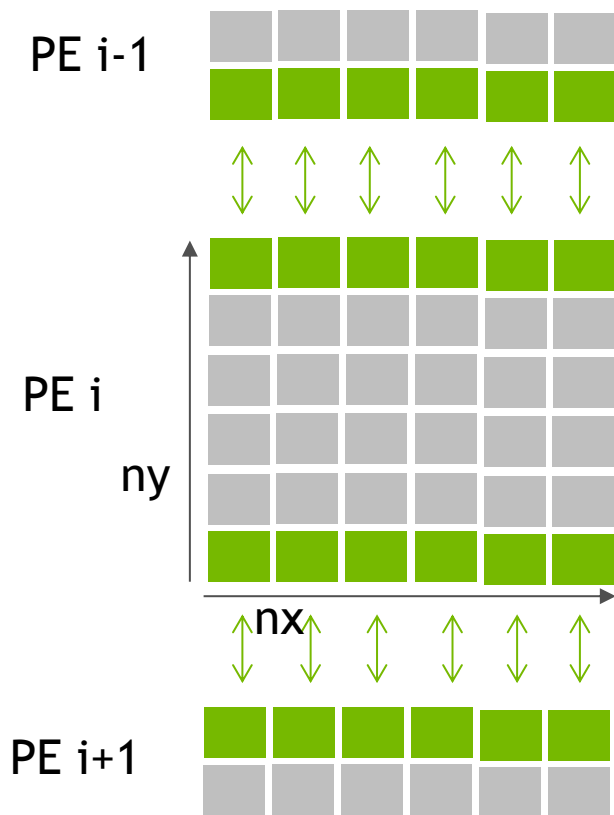
Experimental implementation of OpenSHMEM for NVIDIA GPUs, 1 PE/GPU

shared memory: `shmem_malloc`

private memory: `cudaMalloc`

shmem communication APIs: `shared->shared` or `private->shared`

DEVICE-INITIATED COMMUNICATION



Thread-level communication APIs

Allow finer grained control and overlap

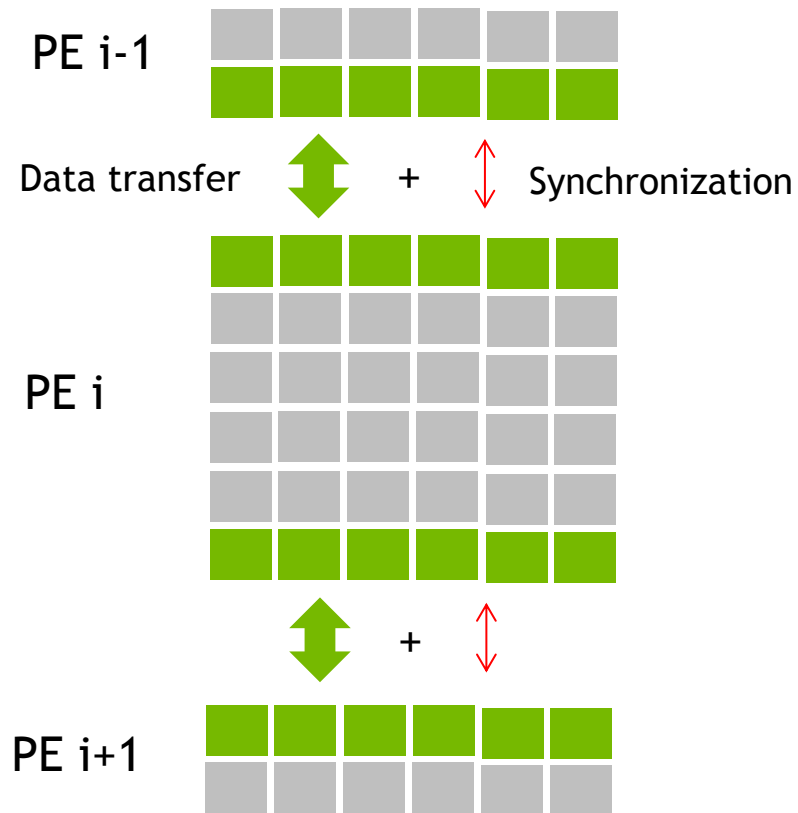
Maps well onto NVLink fabric – DGX-1/DGX-2

```
__global__ void stencil_single_step (float *u, ...)
{
    int ix = threadIdx.x, iy = threadIdx.y;

    //compute

    //data exchange
    if (iy == ny) {
        shmem_float_p (u + ny*nx + ix, u + ix, top_pe);
    }
    if (iy == 1) {
        shmem_float_p (u + nx + ix, u + (ny+1)*nx + ix, bottom_pe);
    }
}
```

IN-KERNEL SYNCHRONIZATION



Allows inter-kernel synchronization

Can offload longer running and cooperating CUDA kernels

```
__global__ void stencil_uber (u, ...)
{
    while (iter=0; iter<N; iter++) {
        //compute

        //data exchange
        shmem_float_put_block_nbi (u + ny*nx, u, nx, top_pe);

        shmem_float_put_block_nbi (u + nx, u + (ny+1)*nx, nx,
                                   bottom_pe);

        shmem_barrier_all();
    }
}
```


CUDA STREAM ORDERED EXTENSIONS

Not all communication can be moved into a CUDA kernels

also not easy in
OpenACC kernels

Not all compute can be fused in to a single kernel

Synchronization or communication at kernel boundary is still required

Extension to offload CPU-initiated SHMEM operations onto a CUDA stream

Eg:

```
kernel1<<<..., stream>>> (...)
```

```
shmemx_barrier_all_on_stream(stream) //can be a collective or p2p operation
```

```
kernel2<<<..., stream>>> (...)
```

NVSHMEM STATUS

Research vehicle for designing and evaluating GPU-centric workloads

Early access available - apply at <http://develop.nvidia.com/nvshmem>

Main Features

- NVLink and PCIe support

- InfiniBand

- X86 and Power9 support

- Interoperability with MPI and OpenSHMEM libraries

NVSHMEM IMPROVES SCALING

Strong scaling Lattice QCD Wilson operator (QUDA) on DGX-2

