N-WAYS TO MULTI-GPU PROGRAMMING

OpenACC
More Science, Less Programming

OPEN HACKATHONS

MULTI-GPU PROGRAMMING

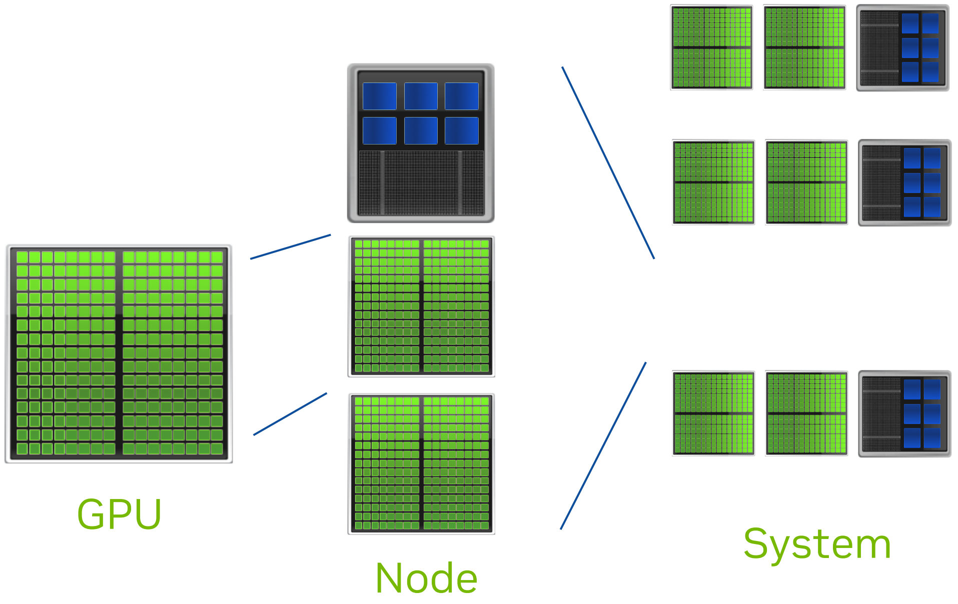
What will we cover?

- Goal: Developing CUDA-aware multi-node multi-GPU applications
- Profiling the application with NVIDIA Nsight Systems
- Communication architecture and system topology
- Optimizations such as overlapping compute and communication
- CUDA concepts like streams and events
- GPUDirect technologies like P2P and RDMA
- Communication libraries: MPI, NVIDIA NCCL and NVSHMEM

Multiple GPUS

Accelerating at all scales

- Unified Memory
- Multi-Process Service – GROMACS blog
- NVLink / NVSwitch and new NVLink Switch!
- CUDA-aware MPI
- NVSHMEM
- NCCL – multi-GPU/node communication primitives
- GPUDirect – comms between GPUs
    - intra- and inter-node
    - now also to storage
- Networking – DPU, SHARP
- Analysis: Nsight tools
- Note: DL Frameworks on NGC and many other HPC applications already have multiGPU and multi-node support built in



APPLICATION OVERVIEW

Solving Laplacian Equation using iterative Jacobi Method

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

Dirichlet boundary conditions on left and right boundaries

Periodic boundary conditions on top and bottom boundaries

Jacobi method pseudocode while the grid has not converged:

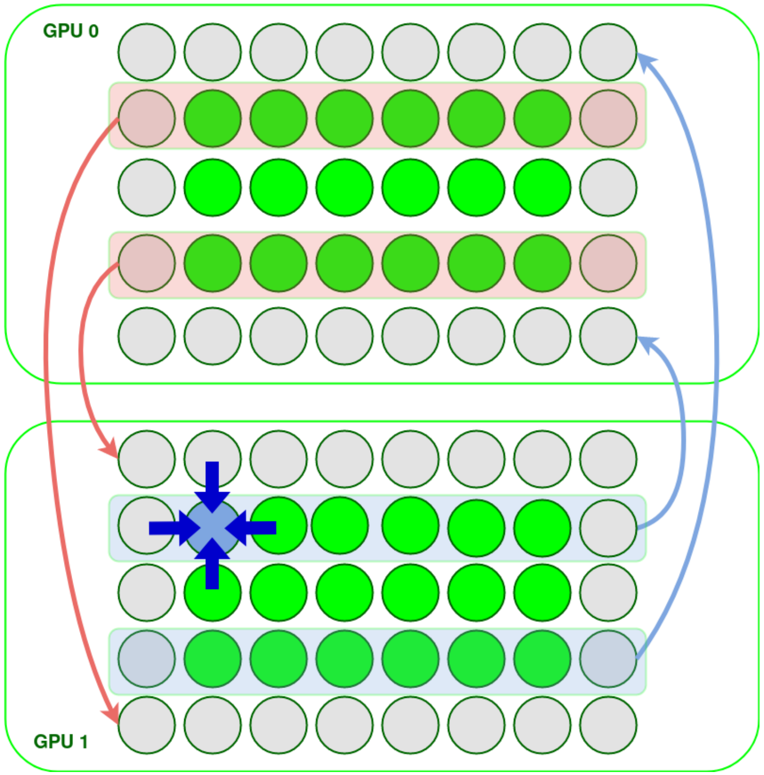
Do Jacobi step:

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



Halo Exchange

DOMAIN DECOMPOSITION

Different Ways to split the work between processes:

Minimize number of neighbors:

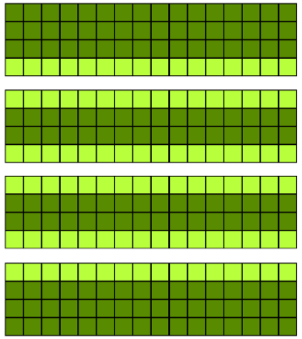
Communicate to less neighbors

Optimal for latency bound communication

Minimize surface area/volume ratio:

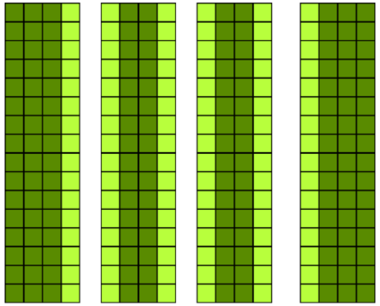
Communicate less data

Optimal for bandwidth bound communication



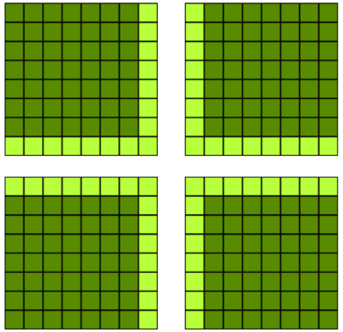
Horizontal Stripes

Contiguous if data is row-major



Vertical Stripes

Contiguous if data is column-major



Tiles

GPUDIRECT FAMILY[1]

Enabling technologies

GPUDIRECT SHARED GPU-SYSMEM

GPU pinned memory shared with other RDMA-capable devices
Avoids intermediate copies

GPUDIRECT P2P

Accelerated GPU-GPU memory copies
Inter-GPU direct load/store access

GPUDIRECT RDMA[2]

Direct GPU to 3rd party device transfers
E.g. direct I/O, optimized inter-node communication

GPUDIRECT ASYNC

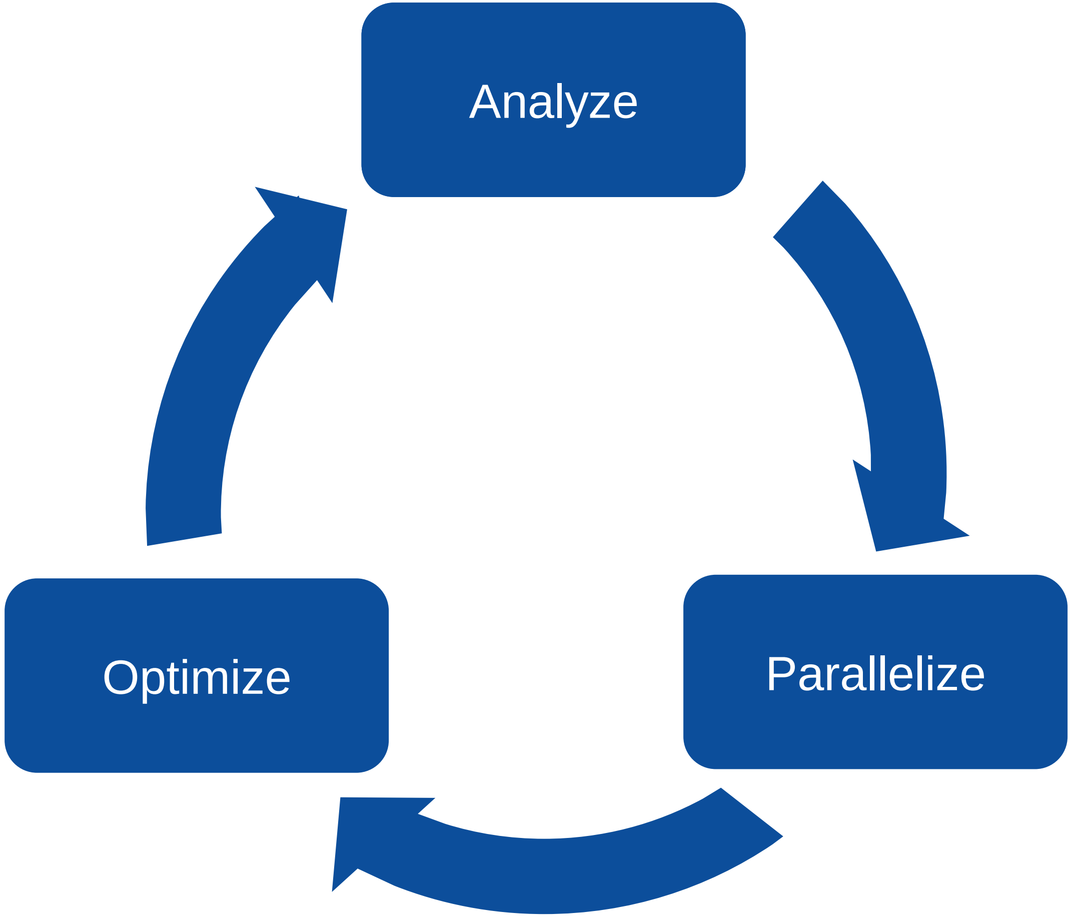
Direct GPU to 3rd party device synchronizations
E.g. optimized inter-node communication

[1] https://developer.nvidia.com/gpudirect

[2] http://docs.nvidia.com/cuda/gpudirect-rdma

DEVELOPMENT CYCLE

- **Analyze** your code to determine most likely places needing parallelization or optimization.
- **Parallelize** your code by starting with the most time consuming parts and check for correctness.
- **Optimize** your code to improve observed speed-up from parallelization.



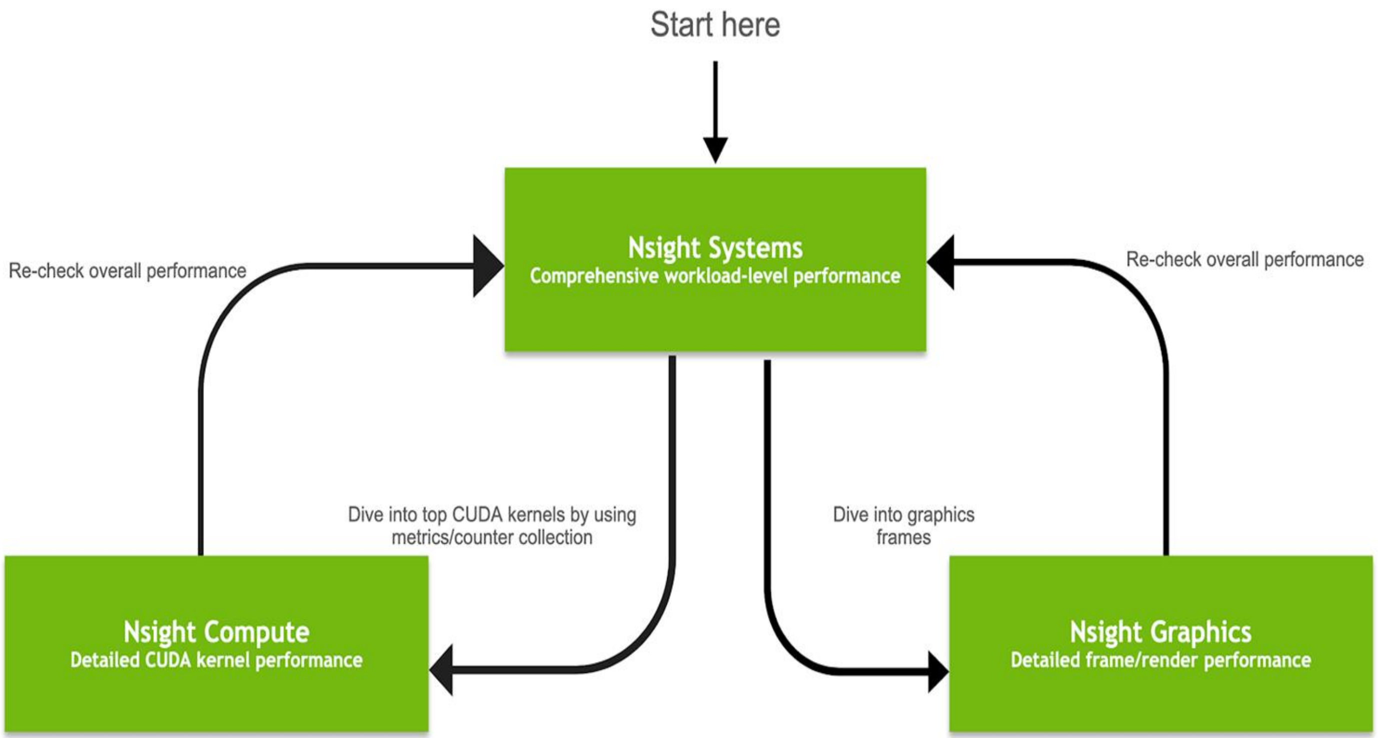
Nsight Product Family

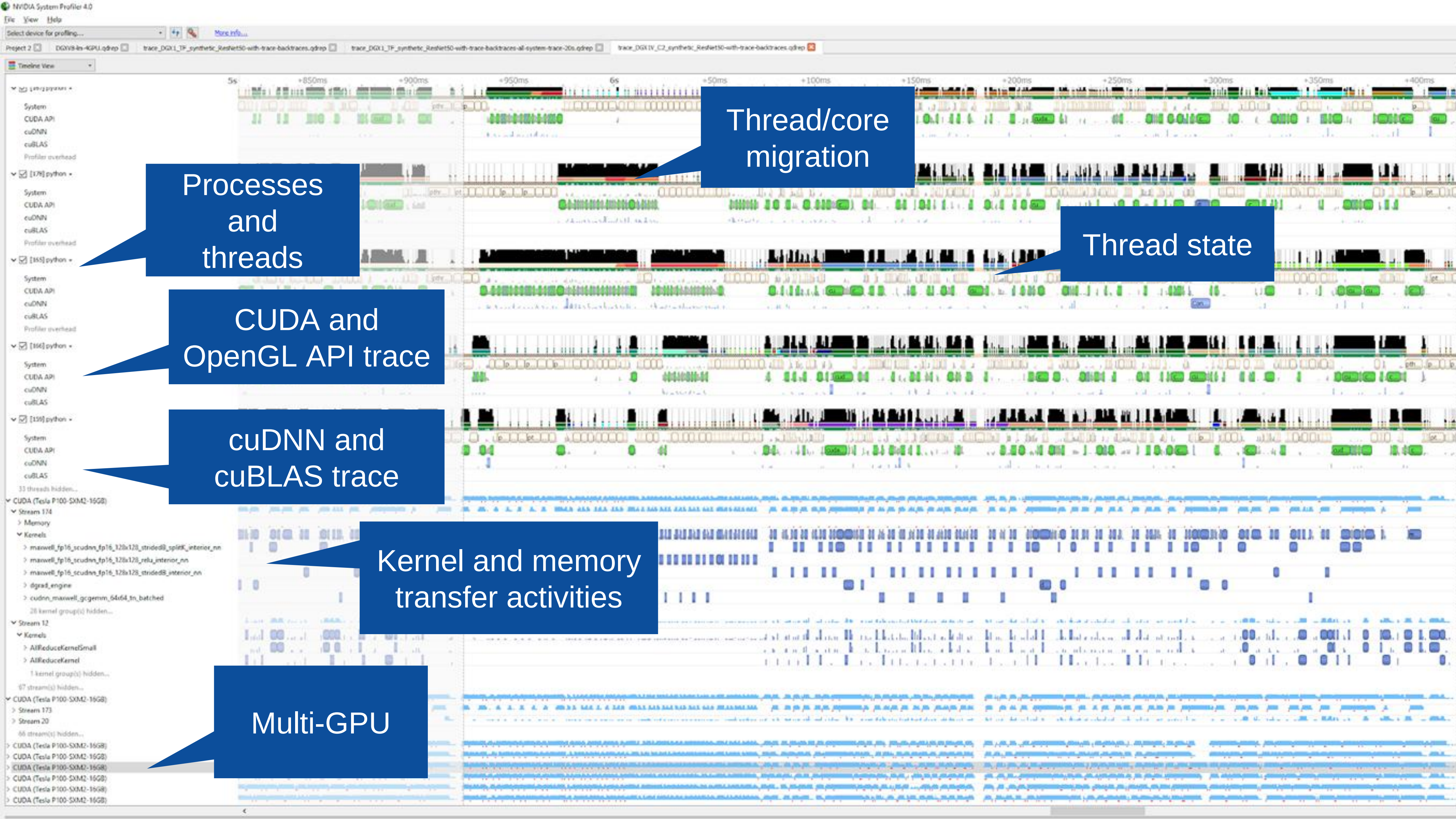
Workflow

Nsight Systems - Analyze application algorithm system-wide

Nsight Compute - Debug/optimize CUDA kernel

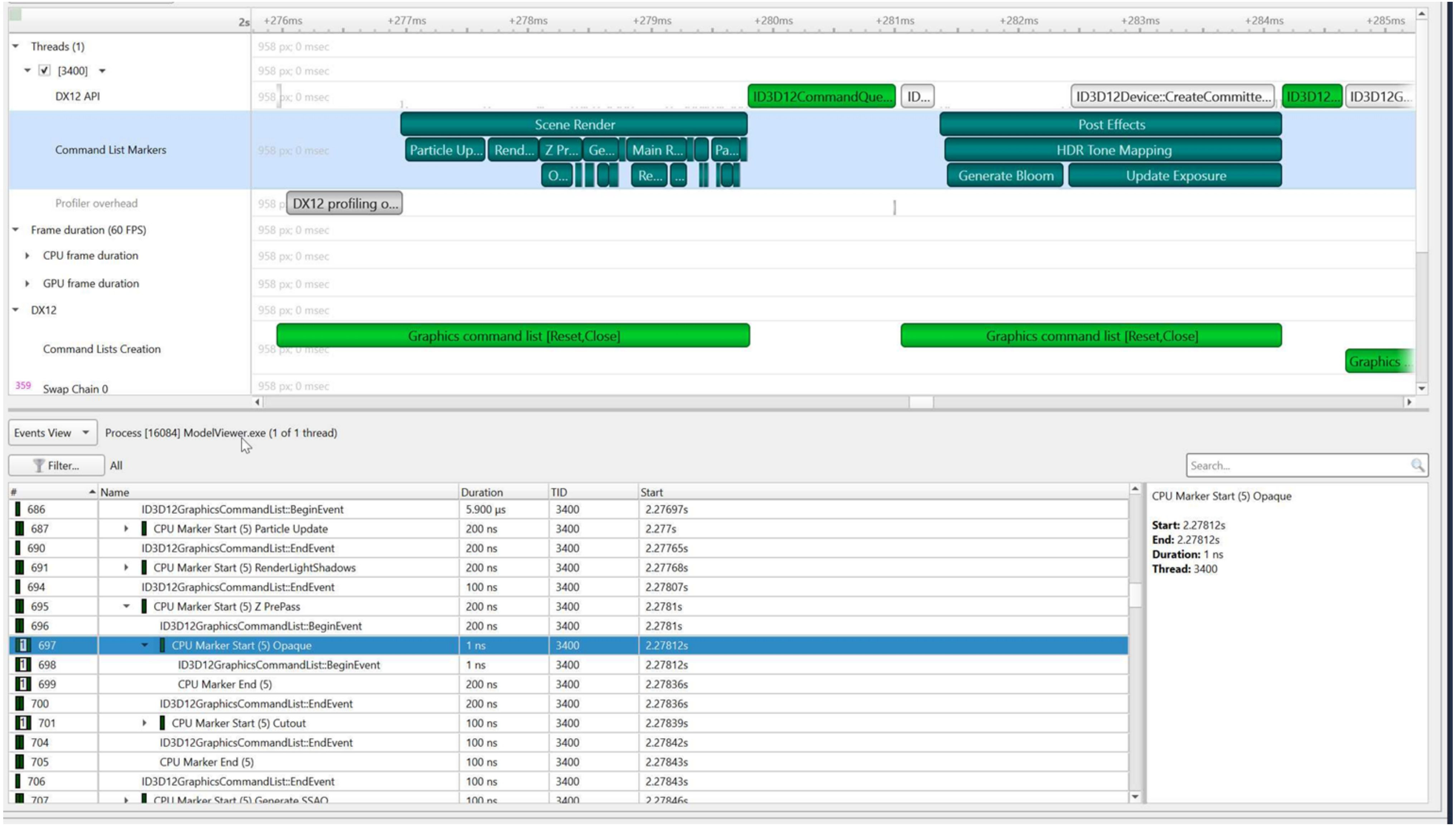
Nsight Graphics - Debug/optimize graphics workloads




NVIDIA System Profiler 4.0
Thread/core migration
Processes and threads
Thread state
CUDA and OpenGL API trace
cuDNN and cuBLAS trace
Kernel and memory transfer activities
Multi-GPU

USER ANNOTATIONS APIS FOR CPU & GPU
NVTX, OPENGL, VULKAN, AND DIRECT3D PERFORMANCE MARKERS

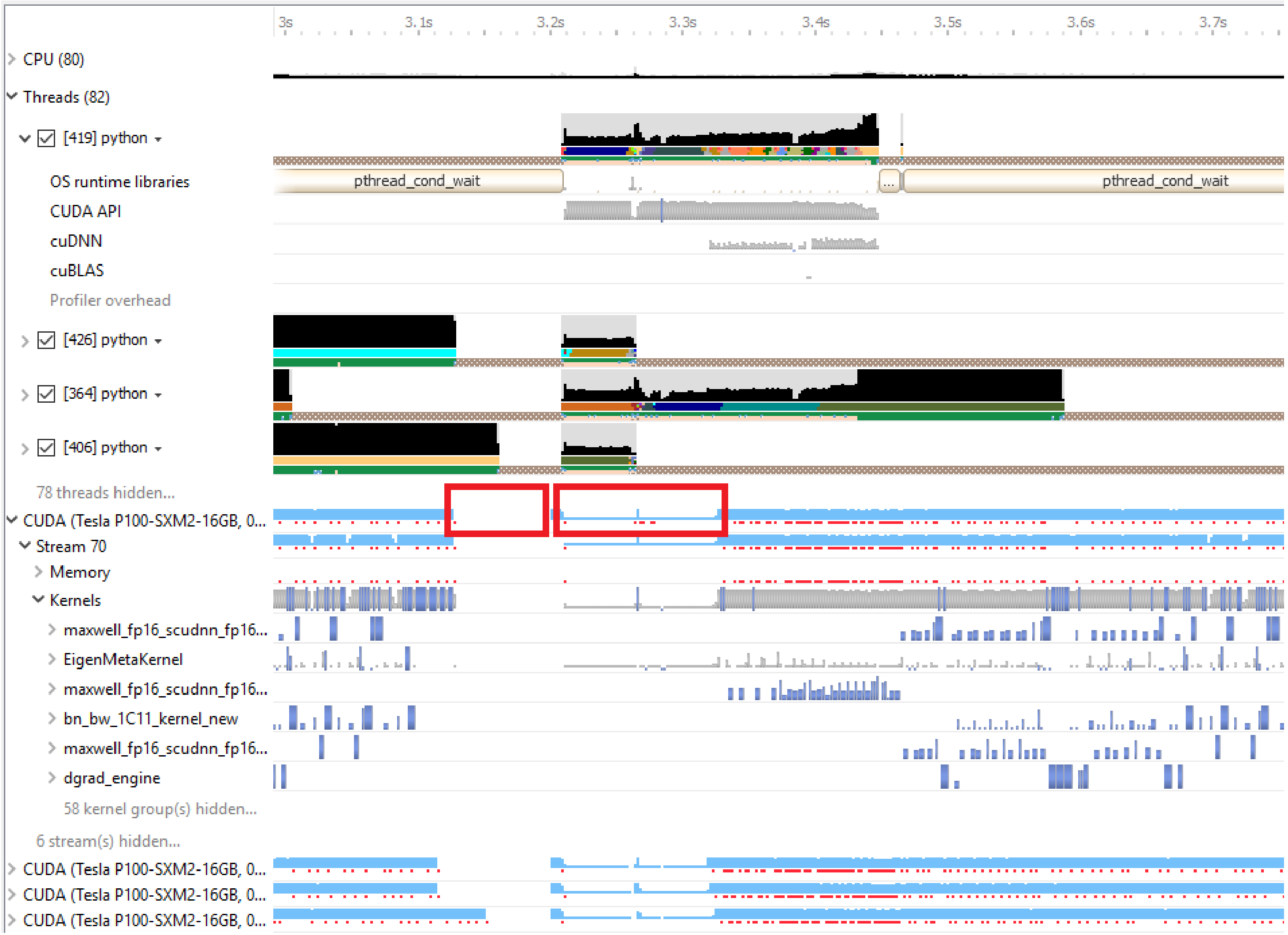
EXAMPLE: VISUAL MOLECULAR DYNAMICS (VMD) ALGORITHMS VISUALIZED WITH NVTX ON CPU



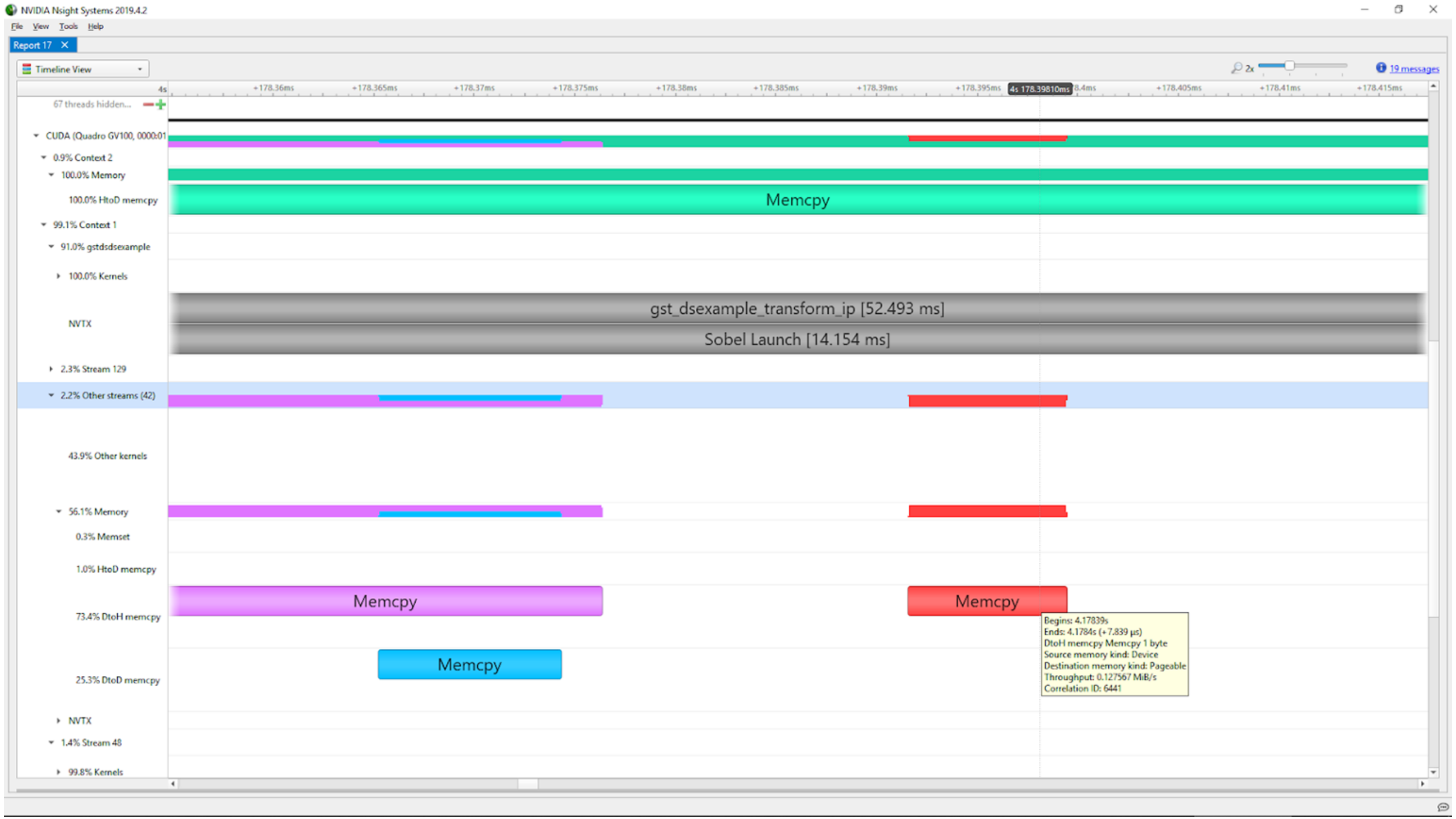
+276ms
+277ms
+278ms
+279ms
+280ms
+281ms
+282ms
+283ms
+284ms
+285ms
Threads (1)
[3400]
DX12 API
Command List Markers
Profiler overhead
Frame duration (60 FPS)
CPU frame duration
GPU frame duration
DX12
Command Lists Creation
Swap Chain 0
ID3D12CommandQue...
ID3D12Device::CreateCommitte...
Scene Render
Post Effects
Particle Up...
Rend...
Z Pr...
Ge...
Main R...
HDR Tone Mapping
Generate Bloom
Update Exposure
DX12 profiling o...
Graphics command list [Reset,Close]
Graphics command list [Reset,Close]
Events View
Process [16084] ModelViewer.exe (1 of 1 thread)
Filter...
All
Search...
| Name | Duration | TID | Start
686 | ID3D12GraphicsCommandList::BeginEvent | 5.900 µs | 3400 | 2.27697s
687 | CPU Marker Start (5) Particle Update | 200 ns | 3400 | 2.277s
690 | ID3D12GraphicsCommandList::EndEvent | 200 ns | 3400 | 2.27765s
691 | CPU Marker Start (5) RenderLightShadows | 200 ns | 3400 | 2.27768s
694 | ID3D12GraphicsCommandList::EndEvent | 100 ns | 3400 | 2.27807s
695 | CPU Marker Start (5) Z PrePass | 200 ns | 3400 | 2.2781s
696 | ID3D12GraphicsCommandList::BeginEvent | 200 ns | 3400 | 2.2781s
697 | CPU Marker Start (5) Opaque | 1 ns | 3400 | 2.27812s
698 | ID3D12GraphicsCommandList::BeginEvent | 1 ns | 3400 | 2.27812s
699 | CPU Marker End (5) | 200 ns | 3400 | 2.27836s
700 | ID3D12GraphicsCommandList::EndEvent | 200 ns | 3400 | 2.27836s
701 | CPU Marker Start (5) Cutout | 100 ns | 3400 | 2.27839s
704 | ID3D12GraphicsCommandList::EndEvent | 100 ns | 3400 | 2.27842s
705 | CPU Marker End (5) | 100 ns | 3400 | 2.27843s
706 | ID3D12GraphicsCommandList::EndEvent | 100 ns | 3400 | 2.27843s
CPU Marker Start (5) Opaque
Start: 2.27812s
End: 2.27812s
Duration: 1 ns
Thread: 3400

EVENT TABLE

MPI & OPENACC TRACE



GPU IDLE AND LOW UTILIZATION LEVEL OF DETAIL



CUDA MEMORY TRANSFER COLOR PALLETTE
SHOW DIRECTION AND PAGEABLE MEMORY HAZARDS

OpenACC More Science, Less Programming

OPEN HACKATHONS

PROFILING SEQUENTIAL CODE

Using Command Line Interface (CLI)

NVIDIA Nsight Systems CLI provides

- Simple interface to collect data
- Can be copied to any system and analysed later
- Profiles both serial and parallel code
- For more info enter `nsys --help` on the terminal

To profile a serial application with NVIDIA Nsight Systems, we use NVIDIA Tools Extension (NVTX) API functions in addition to collecting backtraces while sampling.

PROFILING SEQUENTIAL CODE

NVIDIA Tools Extension API (NVTX) library

What is it?

- A C-based Application Programming Interface (API) for annotating events
- Can be easily integrated to the application
- Can be used with NVIDIA Nsight Systems

Why?

- Allows manual instrumentation of the application
- Allows additional information for profiling (e.g: tracing of CPU events and time ranges)

How?

- Import the header only C library `nvToolsExt.h`
- Wrap the code region or a specific function with `nvtxRangePush()` and `nvtxRangPop(`

```c
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include "laplace2d.h"
#include <nvtx3/nvToolsExt.h>

int main(int argc, char** argv)
{
    const int n = 4096;
    const int m = 4096;
    const int iter_max = 1000;

    const double tol = 1.0e-6;
    double error = 1.0;

    double *restrict A    = (double*)malloc(sizeof(double)*n*m);
    double *restrict Anew = (double*)malloc(sizeof(double)*n*m);

    nvtxRangePushA("init");
    initialize(A, Anew, m, n);
    nvtxRangePop();

    printf("Jacobi relaxation Calculation: %d x %d mesh\n", n, m);

    double st = omp_get_wtime();
    int iter = 0;

    nvtxRangePushA("while");
    while ( error > tol && iter < iter_max )
    {
        nvtxRangePushA("calc");
        error = calcNext(A, Anew, m, n);
        nvtxRangePop();

        nvtxRangePushA("swap");
        swap(A, Anew, m, n);
        nvtxRangePop();

        if(iter % 100 == 0) printf("%5d, %0.6f\n", iter, error);

        iter++;
    }
    nvtxRangePop();

    double runtime = omp_get_wtime() - st;

    printf(" total: %f s\n", runtime);

    deallocate(A, Anew);

    return 0;
}
```

jacobi.c
(starting and ending of ranges are **highlighted** with the **same** color)

-t	Selects the APIs to be traced (nvtx in this example)
--status	if true, generates summary of statistics after the collection
-b	Selects the backtrace method to use while sampling. The option dwarf uses DWARF's CFI (Call Frame Information).
--force-overwrite	if true, overwrites the existing results
-o	sets the output (qdrep) filename

```
mozhgank@prm-dgx-32:~/Code/openacc-training-materials/labs/module4/English/C/solutions/parallel$ nsys profile -t nvtx --stats=true -b dwarf --force-overwrite true -o laplace-seq ./laplace-seq
Collecting data...
Jacobi relaxation Calculation: 4096 x 4096 mesh
    0, 0.250000
  100, 0.002397
  200, 0.001204
  300, 0.000804
  400, 0.000603
  500, 0.000483
  600, 0.000403
  700, 0.000345
  800, 0.000302
  900, 0.000269
 total: 55.754501 s
Processing events...
Capturing symbol files...
Saving intermediate "/home/mozhgank/Code/openacc-training-materials/labs/module4/English/C/solutions/parallel/laplace-seq.qdstrm" file to disk...

Importing [==================================================================100%]
Saved report file to "/home/mozhgank/Code/openacc-training-materials/labs/module4/English/C/solutions/parallel/laplace-seq.qdrep"
Exporting 70802 events: [=================================================100%]

Exported successfully to
/home/mozhgank/Code/openacc-training-materials/labs/module4/English/C/solutions/parallel/laplace-seq.sqlite

Generating NVTX Push-Pop Range Statistics...
NVTX Push-Pop Range Statistics (nanoseconds)

Time(%)   Total Time   Instances      Average       Minimum       Maximum   Range
-------  -----------  ----------  -------------  -------------  -------------  --------
   49.9  55754497966           1  55754497966.0   55754497966   55754497966  while
   26.5  29577817696        1000     29577817.7      29092956      65008545  calc
   23.4  26163892482        1000     26163892.5      25761418      60129514  swap
    0.1    137489808           1    137489808.0     137489808     137489808  init
```

NVTX range statistics

“calc” region (calcNext function) takes 26.6%
“swap” region (swap function) takes 23.4% of total execution time

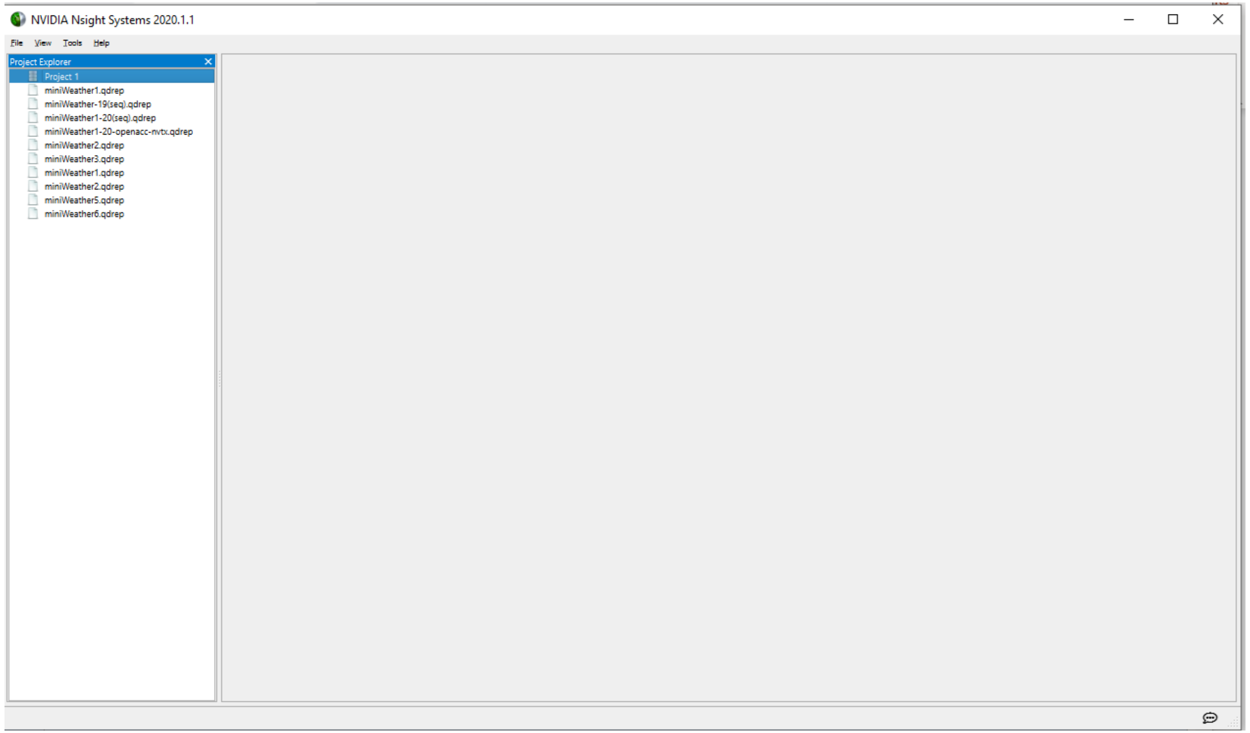
Open laplace-seq.qdrep with Nsight System GUI to view the timeline

OpenACC More Science, Less Programming
OPEN HACKATHONS

PROFILING CODE

Open the generated report files (*.qdrep) from command line in the Nsight Systems profiler.

File > Open



PROFILING CODE

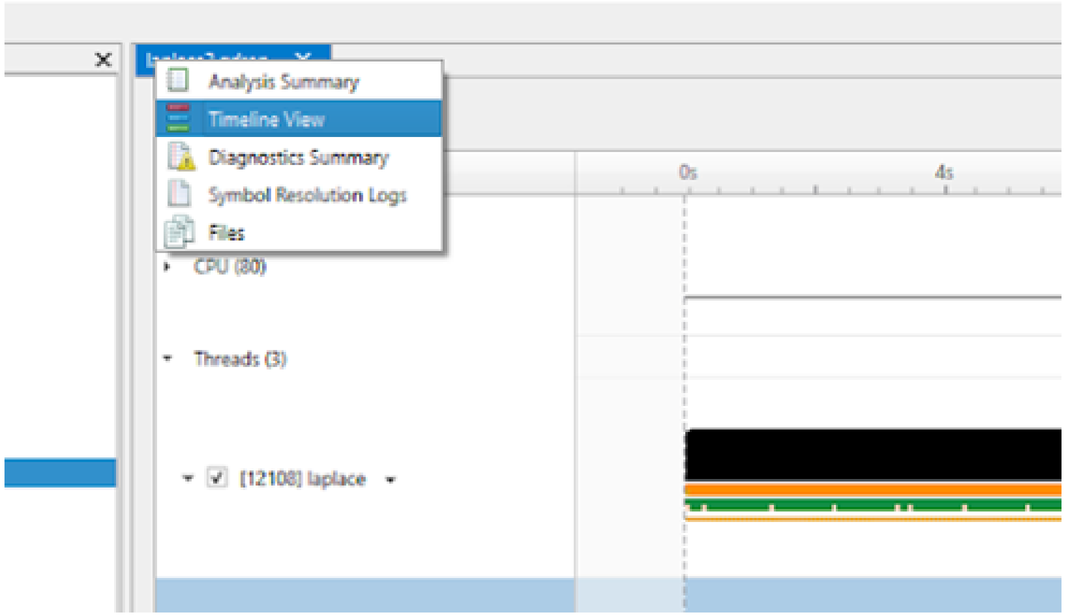
Using Nsight Systems

Navigate through the “view selector”.

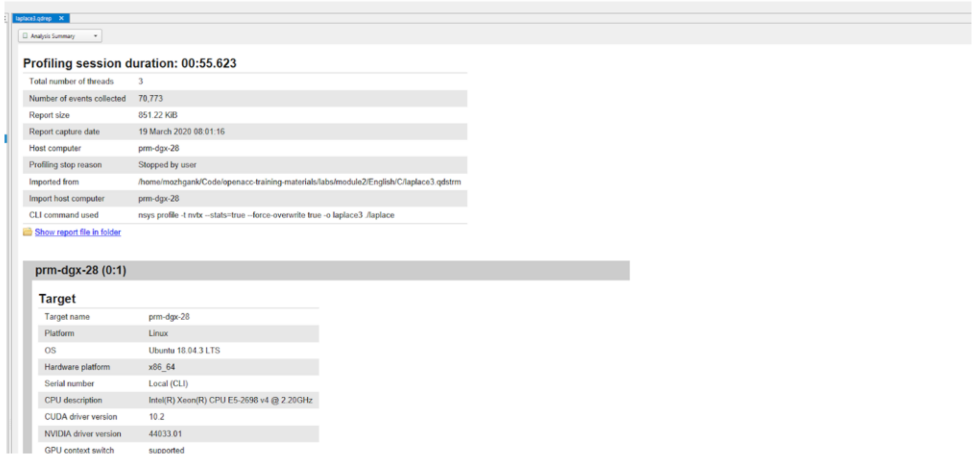
“Analysis summary” shows a summary of the profiling session. To review the project configuration used to generate this report, see next slide.

“Timeline View” contains the timeline at the top, and a bottom pane that contains the events view and the function table.

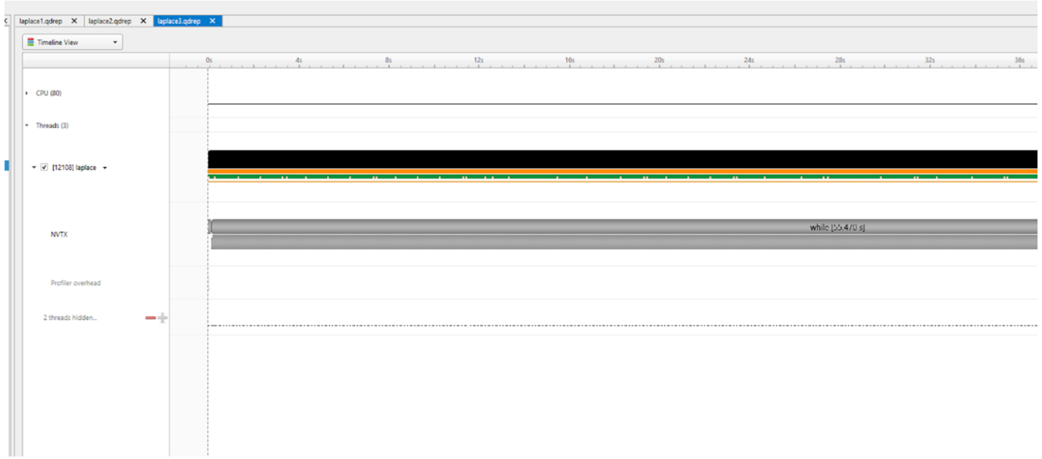
Read more: https://docs.nvidia.com/nsight-systems



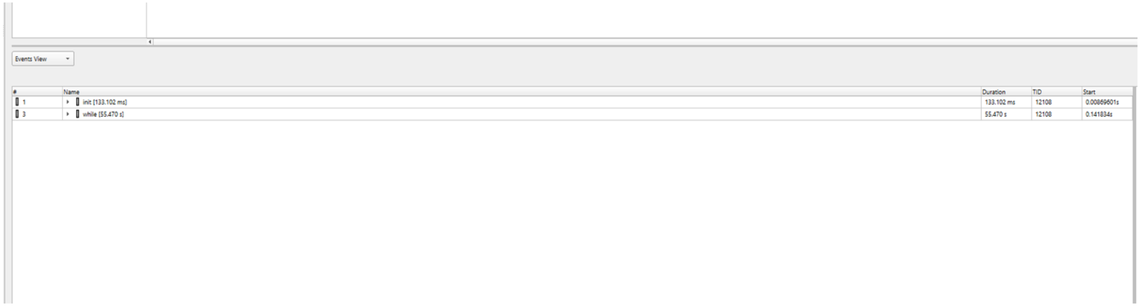
PROFILING CODE



Analysis Summary



Timeline view
(charts and the hierarchy on the top pane)



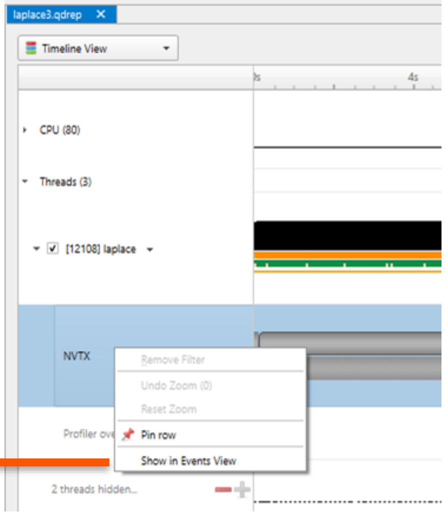
Timeline view
(event view and function table on the bottom pane)

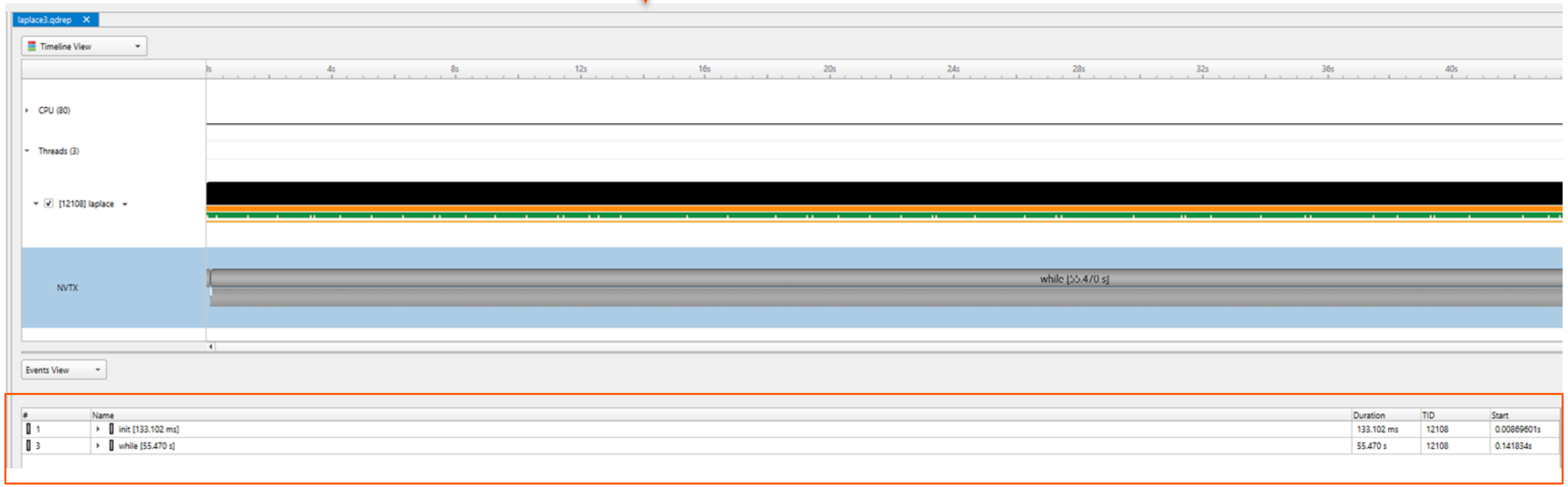
PROFILING CODE

Viewing captured NVTX events and time

From the Timeline view, right click on the “NVTX” from the top pane and choose “Show in Events View”.

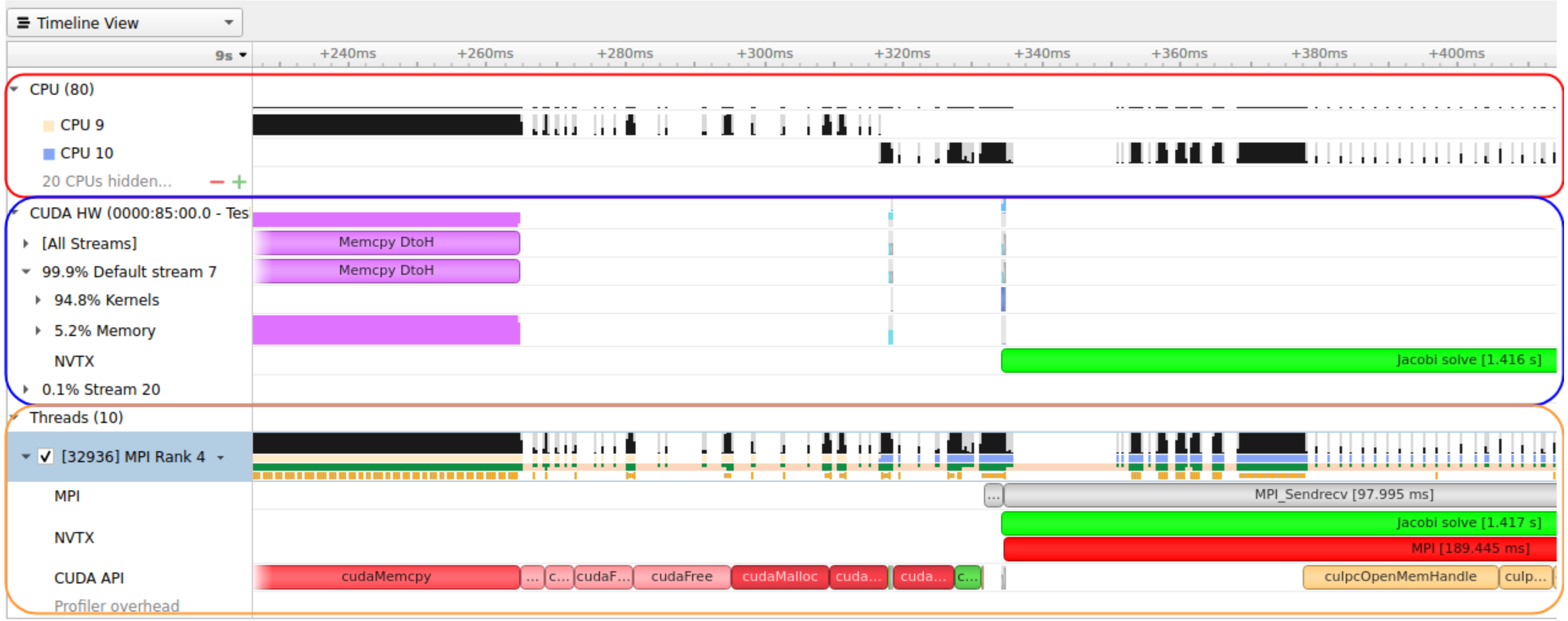
From the bottom pane, you can now see name of the events captured with the ration.





PROFILING: NVIDIA NSIGHT SYSTEMS

```
$ nsys profile --trace=cuda,nvtx --stats=true -o <report_name> --force-overwrite true <program>
```



- CPU tab: thread-level core utilization data.
- CUDA HW tab: GPU kernel and memory transfer activities.
- Threads tab: each CPU thread's activity like CUDA, MPI, NVTX, etc.

Single node Multi-GPU

OpenACC
More Science, Less Programming

OPEN HACKATHONS

SINGLE-NODE MULTI-GPU: CUDA MEMCPY

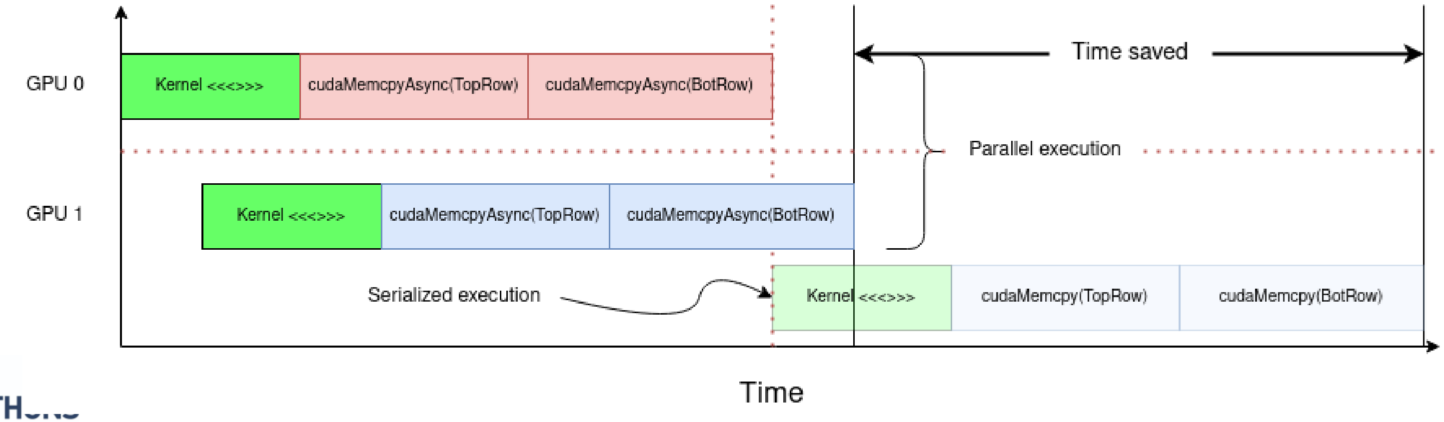
Multi-GPU Jacobi solver:

- Set current device
- Launch device kernel
- Asynchronously copy top halo
- Asynchronously copy bottom halo
- Check norm and swap grid arrays

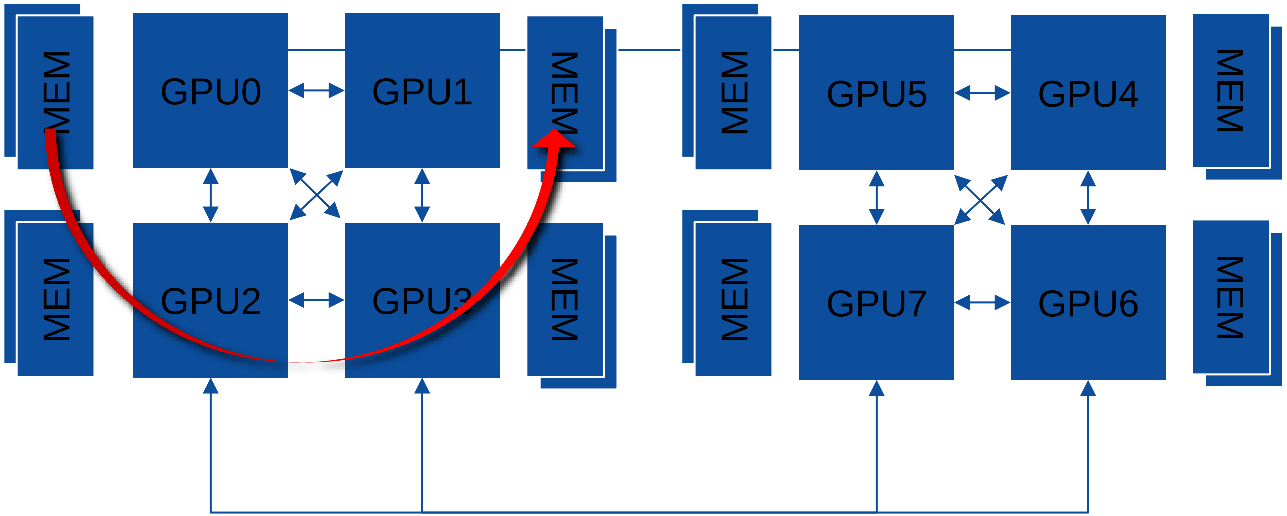
Pseudocode

```
for (int i = 0; i < 2; i++) {
    // Set current device
    cudaSetDevice(i);
    // Launch device kernel on GPU i
    jacobi_kernel<<<dim_grid, dim_block>>>(...);
}
for (int i = 0; i < 2; i++) {
    // Define row number of top and bottom neighbours, etc.
    TopNeighbour = ...; BotNeighbour = ...; // and so-on
    // Halo exchange, notice the use of Async function
    cudaMemcpyAsync(grid_rows[TopNeighbour], grid_rows[myTop], size, cudaMemcpyDeviceToDevice);
    cudaMemcpyAsync(grid_rows[BotNeighbour], grid_rows[myBot], size, cudaMemcpyDeviceToDevice);
    // Norm check, swapping current and previous grid arrays, etc.
} // Parallel execution across multiple GPUs
```

Asynchronous copy timeline



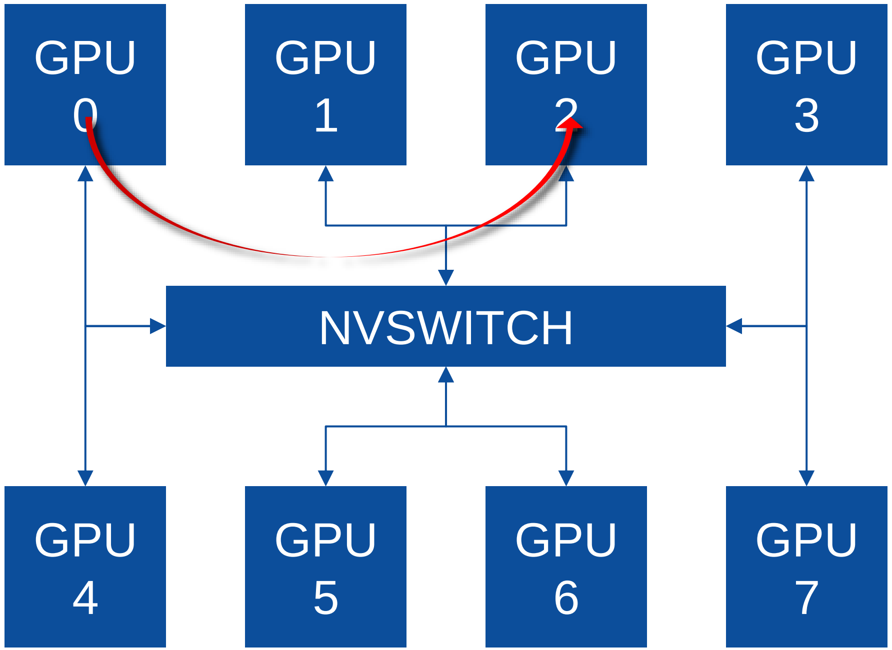
GPUDIRECT P2P



Maximizes intra node inter GPU Bandwidth

Avoids Host memory and system topology bottlenecks

GPUDIRECT P2P



Maximizes intra node inter GPU Bandwidth

Avoids Host memory and system topology bottlenecks

SINGLE-NODE MULTI-GPU: P2P

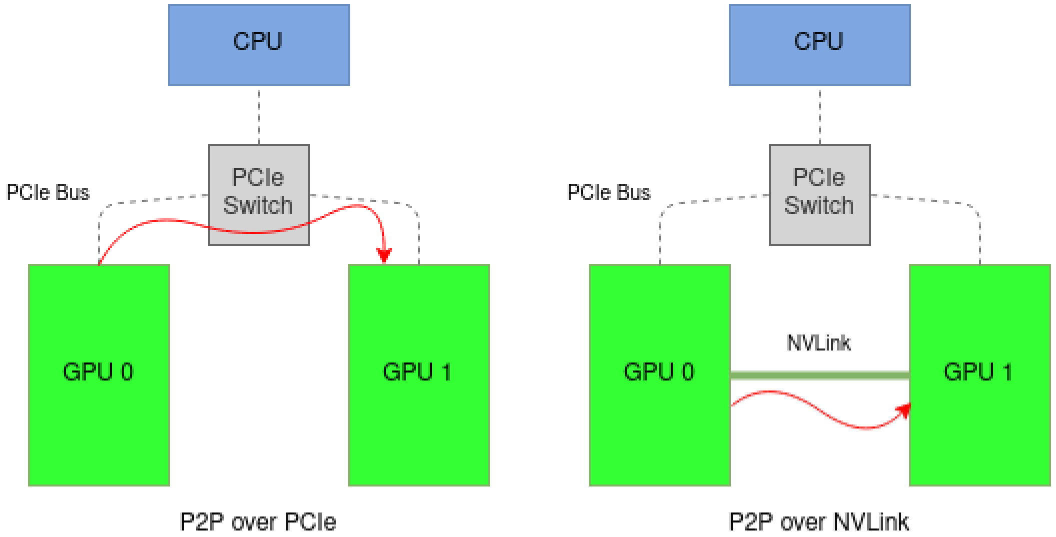
Host staging increases latency and decreases bandwidth

Peer-to-Peer (P2P) Memory Access bypasses host staging and utilizes NVLink and NVSwitch

CLI to check P2P support:

```
cudaSetDevice(currDevice);
int canAccessPeer = 0;
cudaDeviceCanAccessPeer(&canAccessPeer, currDevice, PeerDevice);
if (canAccessPeer) {
    cudaDeviceEnablePeerAccess(PeerDevice, 0);
}
```

```
# Check P2P capability
$ nvidia-smi topo -p2p r
# Check NVLink-based P2P support
$ nvidia-smi topo -p2p n
```



Peer-to-Peer Memory Access bypassing Host Staging

INTRA-NODE TOPOLOGY

CLI:

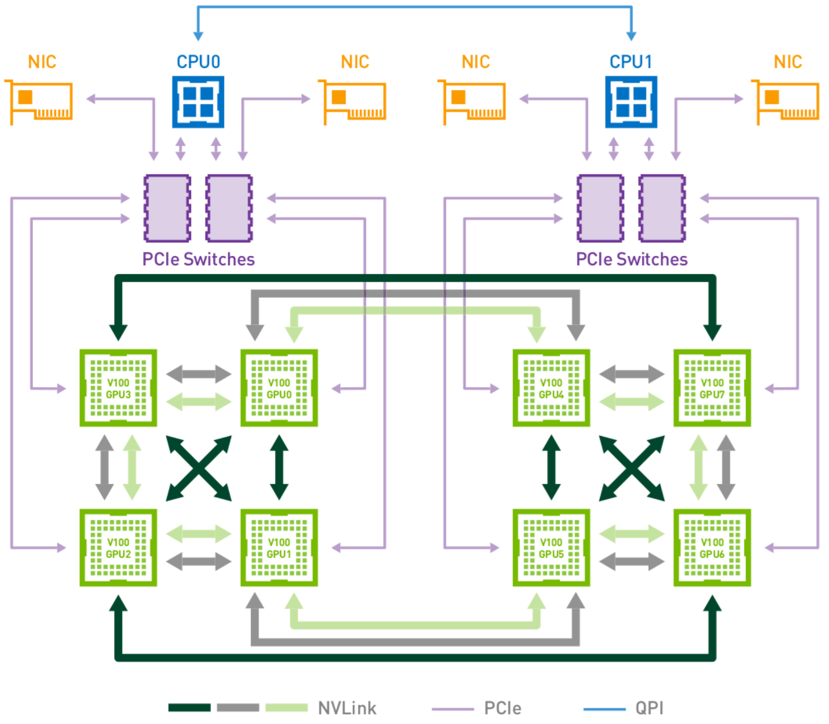
```
$ nvidia-smi topo -m
```

Partial Output:

```
        GPU0    GPU1    GPU2    GPU3    GPU4    GPU5    GPU6    GPU7    mlx5_0  mlx5_1  mlx5_2  mlx5_3  CPU Affinity    NUMA Affinity
GPU0     X      NV1     NV1     NV2     NV2     SYS     SYS     SYS     PIX     PHB     SYS     SYS     0-19,40-59      0
```

Connection Type	Bandwidth	Latency
Double NVLink (NV2)	Very High	Low
Single NVLink (NV1)	High	Low
Single PCIe bridge (PIX)	Medium	Medium
Multiple PCIe bridges (PXB/ PHB)	Low	High
CPU interconnect (NODE/ SYS)	Low	Very High

Bandwidth and Latency of different connections



DGX-1 8 Tesla V100 topology

CUDA STREAMS

SYNCHRONICITY IN CUDA

All CUDA calls are either synchronous or asynchronous w.r.t the host

- Synchronous: enqueue work and wait for completion
- Asynchronous: enqueue work and return immediately
    - Kernel Launches are asynchronous - Automatic overlap with host
    - cudaDeviceSynchronize() makes host wait for GPU operations to finish

CUDA STREAMS

SYNCHRONICITY IN CUDA

- **A stream is a queue of device work**
  - The host places work in the queue and continues on immediately
  - The device schedules work from streams when resources are free
- CUDA operations are placed within a stream — e.g. Kernel launches, memory copies
  - If no stream is specified, the default stream is used
- Operations within the same stream are ordered (FIFO) and cannot overlap
- Operations in different streams are unordered and ***can overlap***
  - ** NB default stream is special case – it is blocking for other streams, so no overlap **

OpenACC More Science, Less Programming

OPEN HACKATHONS

CUDA STREAMS

API

MANAGING STREAMS

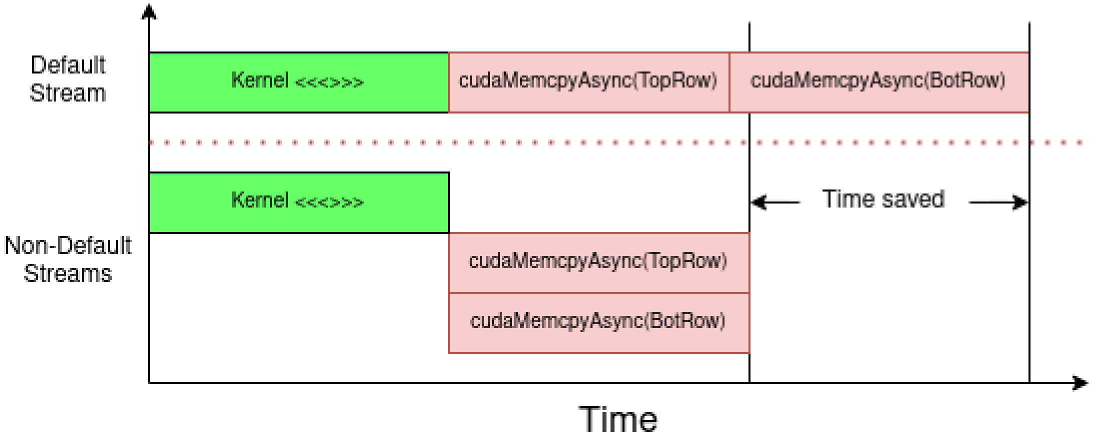
- cudaStream_t stream; - Declares a stream handle
- cudaStreamCreate(&stream); - Allocates a stream
- cudaStreamDestroy(stream); - Deallocates a stream
- cudaStreamSynchronize(stream); - Blocks host progress until work in stream has completed

```
kernel<<< blocks , threads, smem, stream>>>();

cudaMemcpyAsync( dst, src, size, dir, stream);
```

OPTIMIZATIONS: CUDA STREAMS AND EVENTS

Overlap compute and communication with streams:

- Create computation and communication streams
- Use compute stream for Jacobi computation
- Use communication stream for halo exchanges



Timeline using CUDA streams

Fine-grained inter-stream synchronization using CUDA events:

```
# Create streams
cudaStream_t compute_stream, comm_stream;
cudaStreamCreate(&compute_stream); cudaStreamCreate(&comm_stream);
# Use streams
jacobi_kernel<<<dim_grid, dim_block, 0, compute_stream>>>(...);
cudaMemcpyAsync(TopNeighbour, myTopRow, size,
                         cudaMemcpyDeviceToDevice, comm_stream);
```

Streams pseudocode

```
# Create event
cudaEvent_t event1;
cudaEventCreate(&event1);
# Record an event on stream1
cudaEventRecord(&event1, stream1);
# Make stream2 wait until event1 occurs
cudaStreamWaitEvent(stream2, event1, 0);
```

Multi-node Multi-GPU

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, …

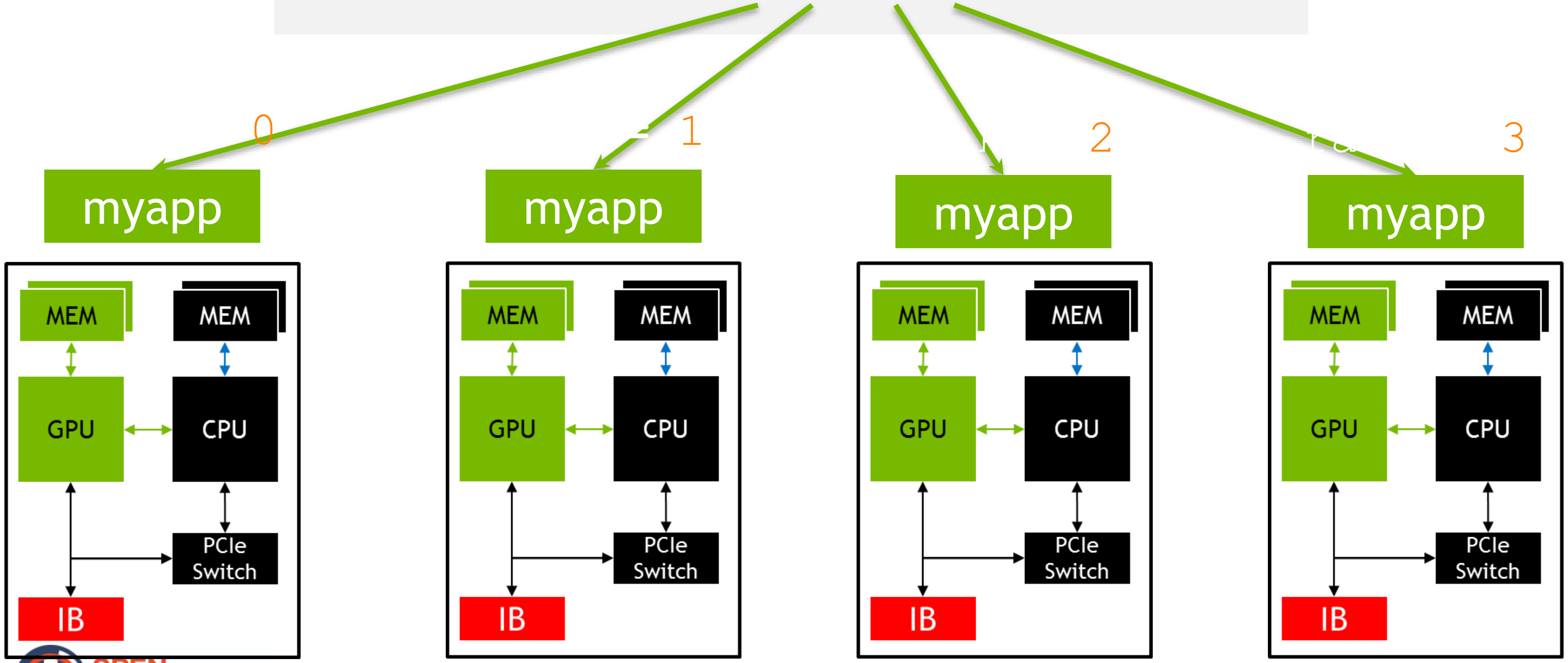
MPI - SKELETON

```
#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;
}
```

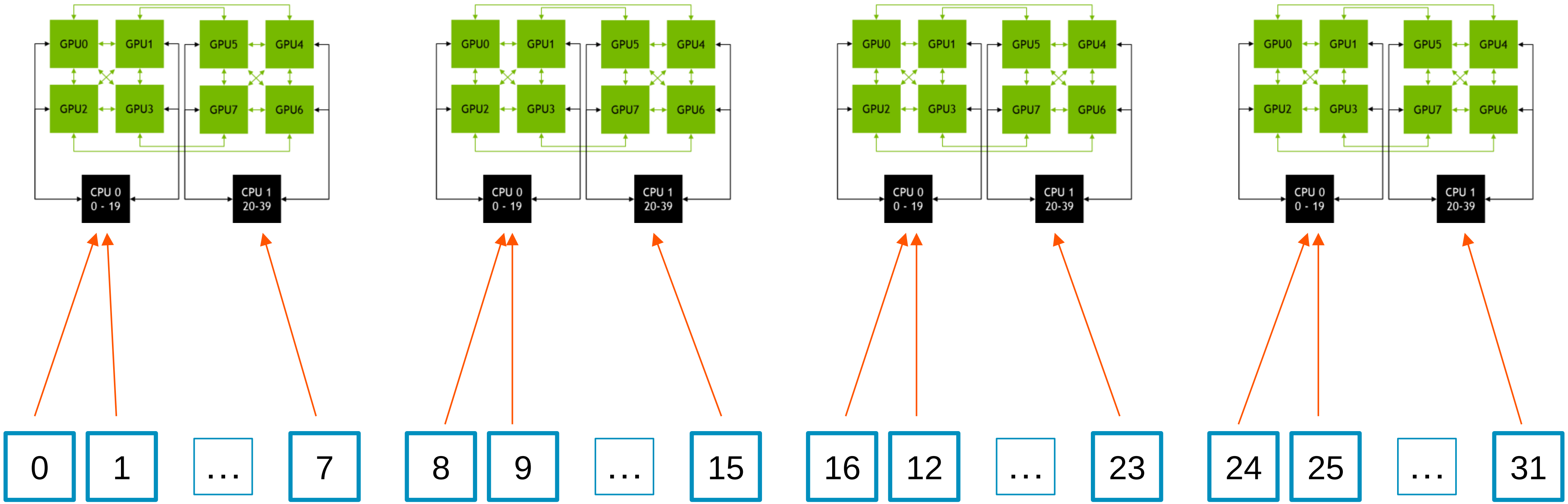
MPI

Compiling and Launching

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



HANDLING MULTIPLE MULTI GPU NODES



HANDLING MULTIPLE MULTI GPU NODES

How to determine the local rank? – MPI-3

```
MPI_Comm local_comm;
MPI_Info info;
MPI_Info_create(&info);
MPI_Comm_split_type(MPI_COMM_WORLD, MPI_COMM_TYPE_SHARED, rank, info, &local_comm);
int local_rank = -1;
MPI_Comm_rank(local_comm,&local_rank);
MPI_Comm_free(&local_comm);
MPI_Info_free(&info);
```

MULTI-NODE MULTI-GPU: MEMCPY + MPI

Multi-node Jacobi solver:

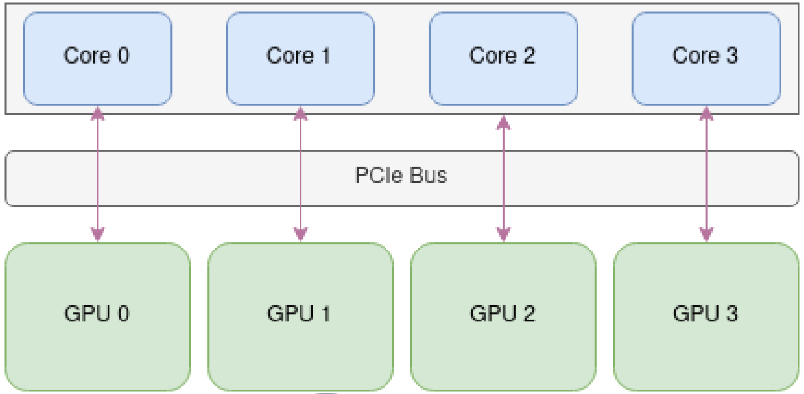
- Set current device and local MPI rank
- Run device kernel and copy halo to host
- Exchange halo with MPI pt-2-pt communication
- Copy new halo back to device
- Check norm with MPI collective communication

```
int MPI_Sendrecv(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
        int dest, int sendtag,
        void *recvbuf, int recvcount, MPI_Datatype recvtype,
        int source, int recvtag,
        MPI_Comm comm, MPI_Status *status);
```

Point-to-Point communication example

```
int MPI_Allreduce(const void *sendbuf, void *recvbuf, int count,
        MPI_Datatype datatype, MPI_Op op, MPI_Comm comm);
```

Collective communication example



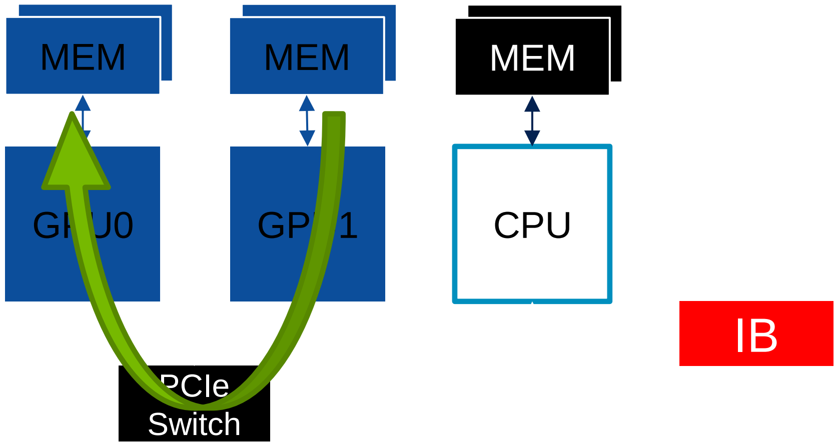
Single rank per GPU communication model

Obtaining node-level local rank:

```
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);
```

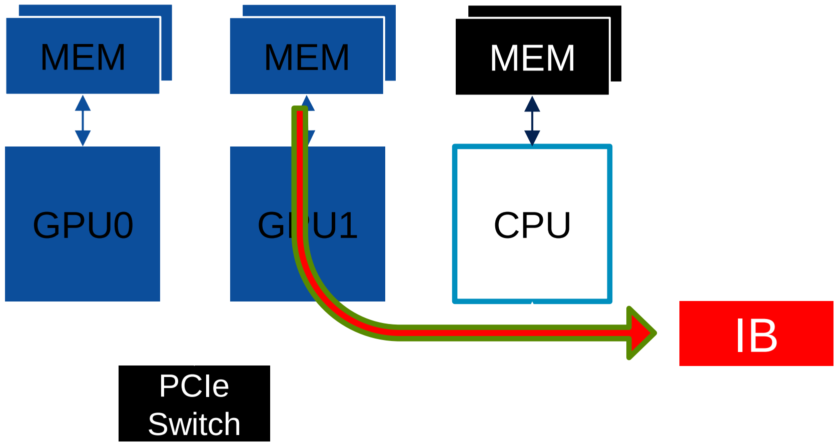
NVIDIA GPUDIRECT™

Peer to Peer Transfers

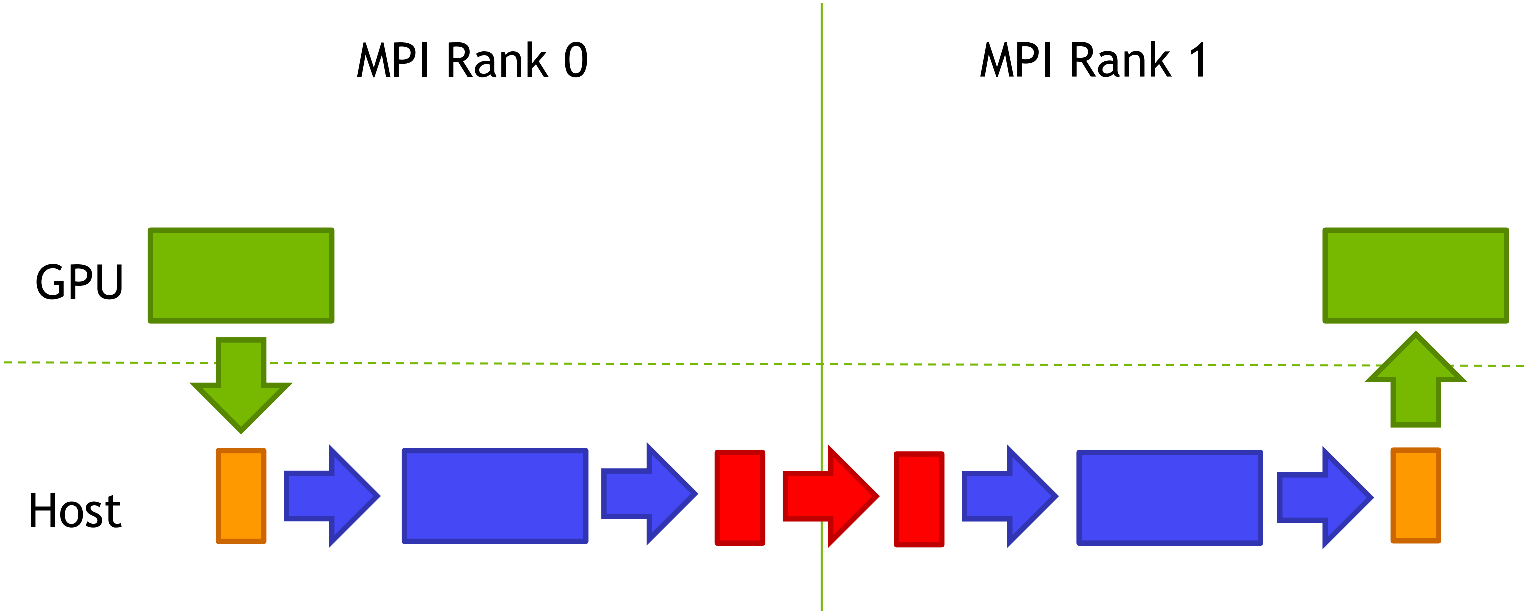


NVIDIA GPUDIRECT™

Support for RDMA

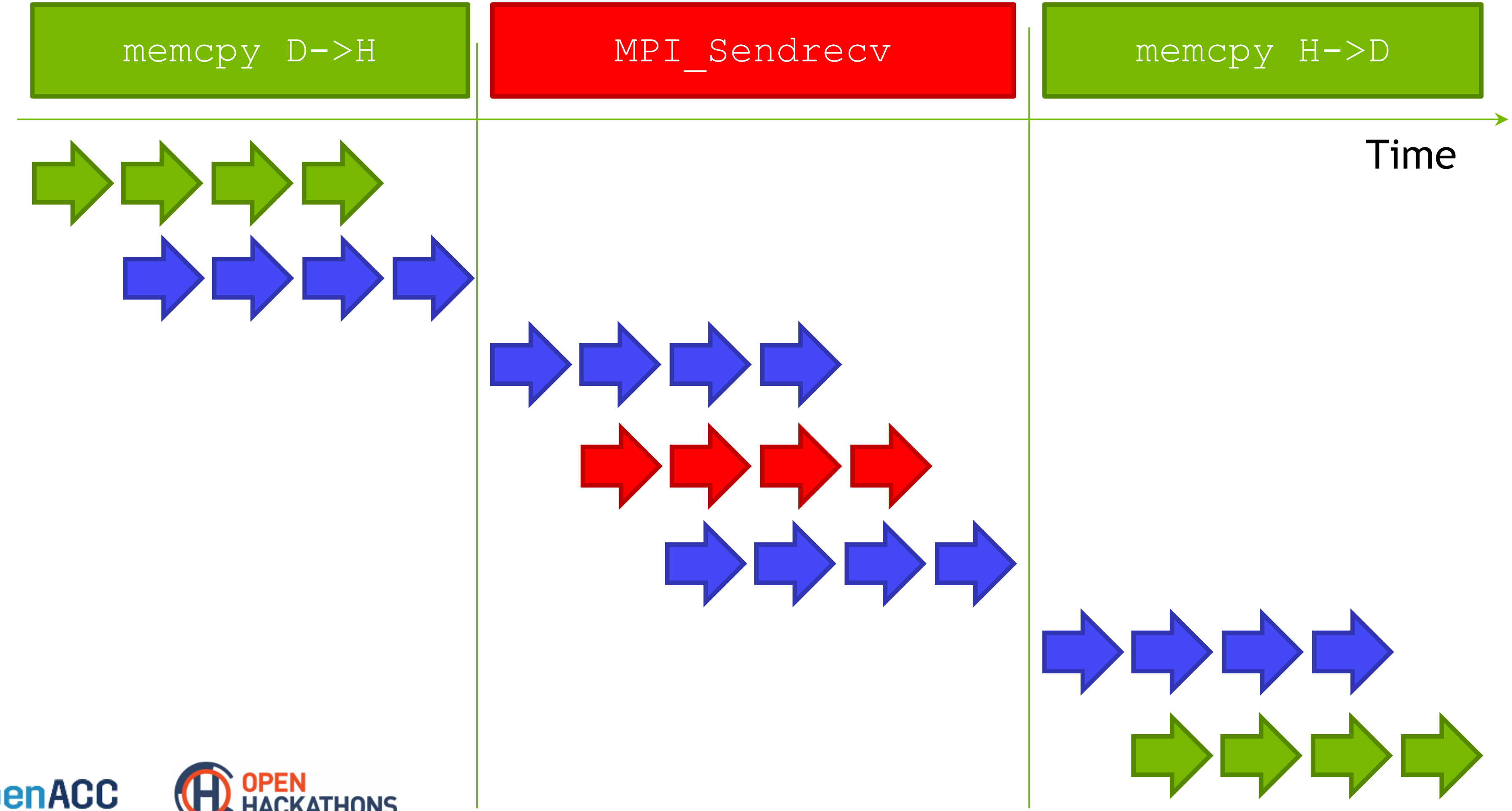


REGULAR MPI GPU TO REMOTE GPU



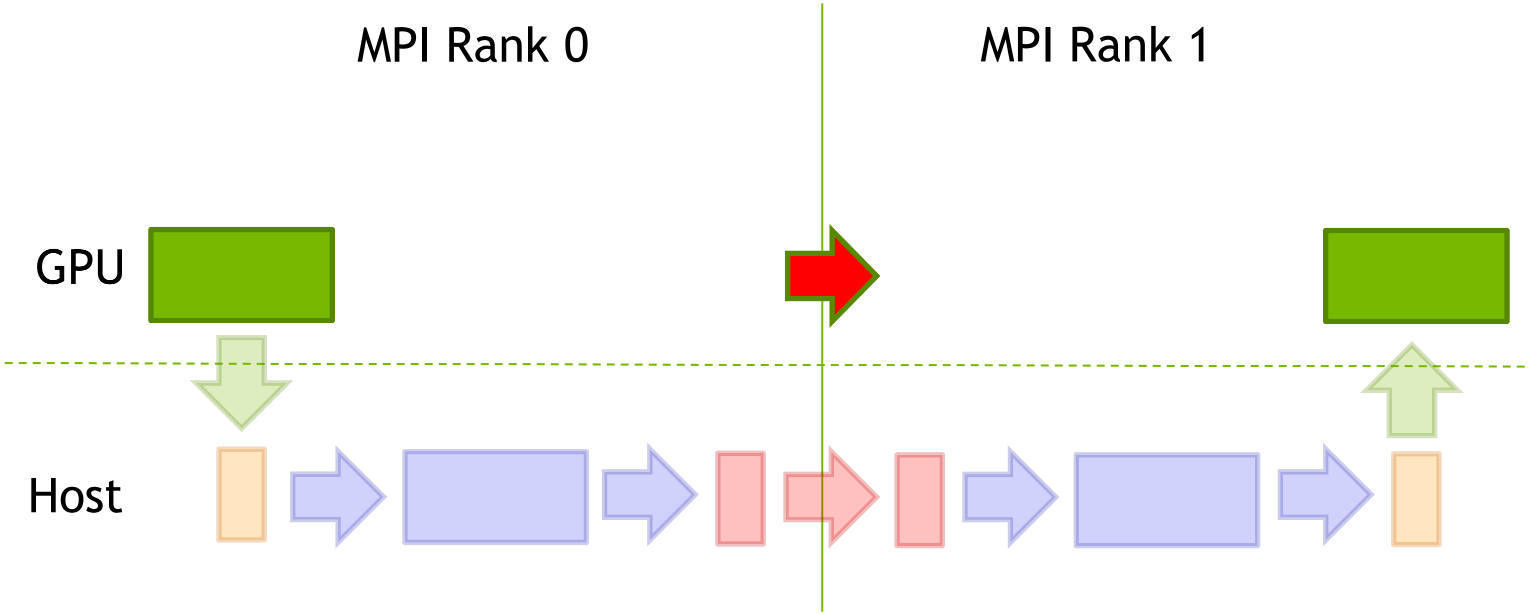

```
cudaMemcpy(s_buf_h,s_buf_d,size,cudaMemcpyDeviceToHost);
MPI_Send(s_buf_h,size,MPI_CHAR,1,tag,MPI_COMM_WORLD);

MPI_Recv(r_buf_h,size,MPI_CHAR,0,tag,MPI_COMM_WORLD,&stat);
cudaMemcpy(r_buf_d,r_buf_h,size,cudaMemcpyHostToDevice);
```

REGULAR MPI GPU TO REMOTE GPU



MPI GPU TO REMOTE GPU

Support for RDMA




```
MPI_Send(s_buf_d,size,MPI_CHAR,1,tag,MPI_COMM_WORLD);

MPI_Recv(r_buf_d,size,MPI_CHAR,0,tag,MPI_COMM_WORLD,&stat);
```

MULTI-NODE MULTI-GPU: CUDA-AWARE MPI

Enables several optimizations and improves ease-of-programming

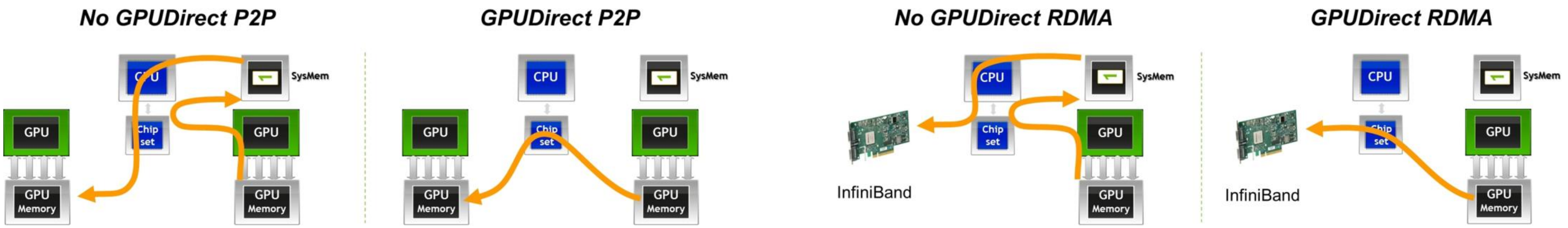
```
//MPI rank 0
cudaMemcpy(s_buf_h, s_buf_d, size, cudaMemcpyDeviceToHost);
MPI_Send(s_buf_h, size, MPI_CHAR, 1, 0, MPI_COMM_WORLD);

//MPI rank n-1
MPI_Recv(r_buf_h, size, MPI_CHAR, 0, 0, MPI_COMM_WORLD, &status);
cudaMemcpy(r_buf_d, r_buf_h, size, cudaMemcpyHostToDevice);
```

Memcpy + MPI

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

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

CUDA-aware MPI

Optimizations applied transparently to the user:



And several more...

NCCL

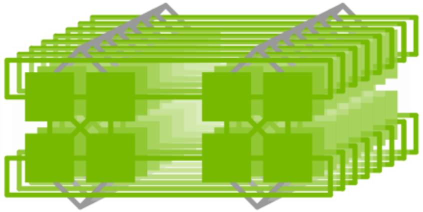
NVIDIA Collective Communications Library (NCCL) 2

Multi-GPU and multi-node collective communication primitives

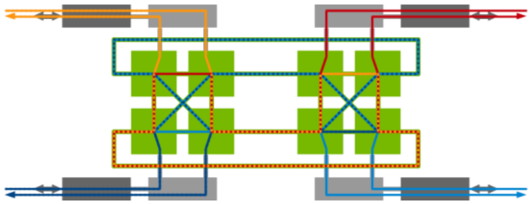
High-performance multi-GPU and multi-node collective communication primitives optimized for NVIDIA GPUs

- Fast routines for multi-GPU multi-node acceleration that maximizes inter-GPU bandwidth utilization
- Easy to integrate and MPI compatible. Uses automatic topology detection to scale HPC and deep learning applications over PCIe and NVlink
- Accelerates leading deep learning frameworks such as Caffe2, Microsoft Cognitive Toolkit, MXNet, PyTorch and more

Multi-GPU:
NVLink
PCIe



Multi-Node:
InfiniBand verbs
IP Sockets



Automatic Topology Detection

developer.nvidia.com/nccl

MULTI-NODE MULTI-GPU: NCCL

Topology-aware communication-centric NVIDIA library

Topology detection

Build graph with all GPUs, NICs, CPUs, PCI switches, NVLink, NVSwitch.

Topology injection for VMs.

Graph search

Extensive search to find optimal set of rings or trees.

Performance prediction of each algorithm and auto-tuning.

Graph connect

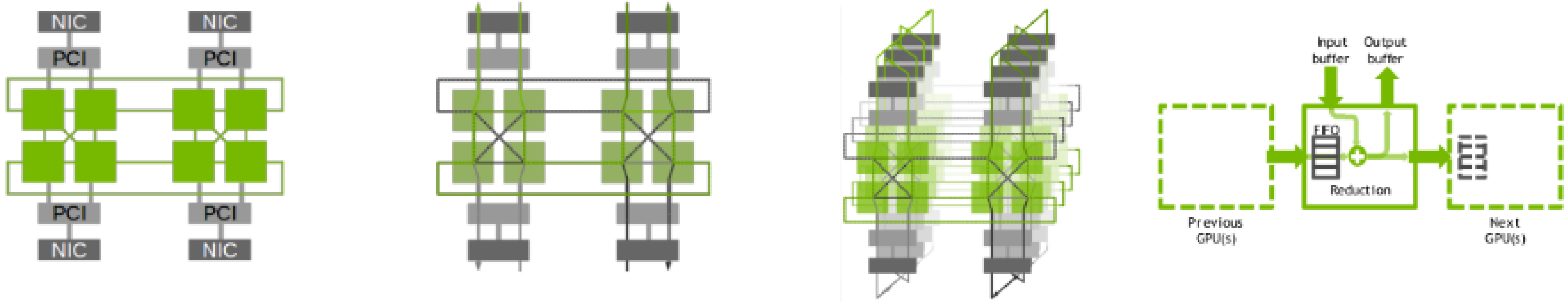
Connect graphs between nodes.

Connect GPUs with intermediate FIFOs, using PCI, NVLink, GPU Direct RDMA, ...

CUDA Kernels

Optimized reductions and copies for a minimal SM usage.

CPU threads for network communication.

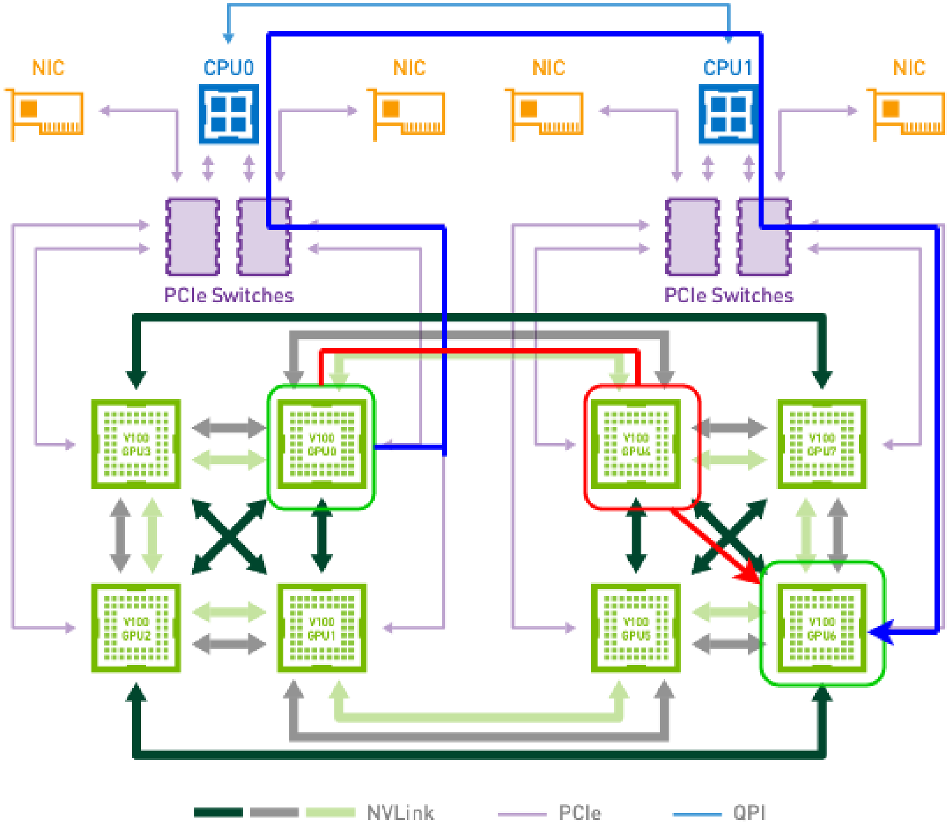


NCCL Architecture

MULTI-NODE MULTI-GPU: NCCL

API:

```
// Communicator creation
ncclGetUniqueId(ncclUniqueId* commId);
ncclCommInitRank(ncclComm_t* comm, int nranks, ncclUniqueId commId,
int rank);

// Communicator destruction
ncclCommDestroy(ncclComm_t comm);

// Point-to-point communication
ncclSend(void* sbuff, size_t count, ncclDataType_t type, int peer,
ncclComm_t comm, cudaStream_t stream);
ncclRecv(void* rbuff, size_t count, ncclDataType_t type, int peer,
ncclComm_t comm, cudaStream_t stream);

// Collective communication
ncclAllReduce(void* sbuff, void* rbuff, size_t count, ncclDataType_t type,
ncclRedOp_t op, ncclComm_t comm, cudaStream_t stream);
ncclBroadcast(void* sbuff, void* rbuff, size_t count, ncclDataType_t
type,       int root, ncclComm_t comm, cudaStream_t stream);

// Aggregation/Composition
ncclGroupStart();
ncclGroupEnd();
```



NCCL optimization: Blue path is taken by CUDA-aware MPI and Red path is taken by NCCL

OpenACC More Science, Less Programming

OPEN HACKATHONS

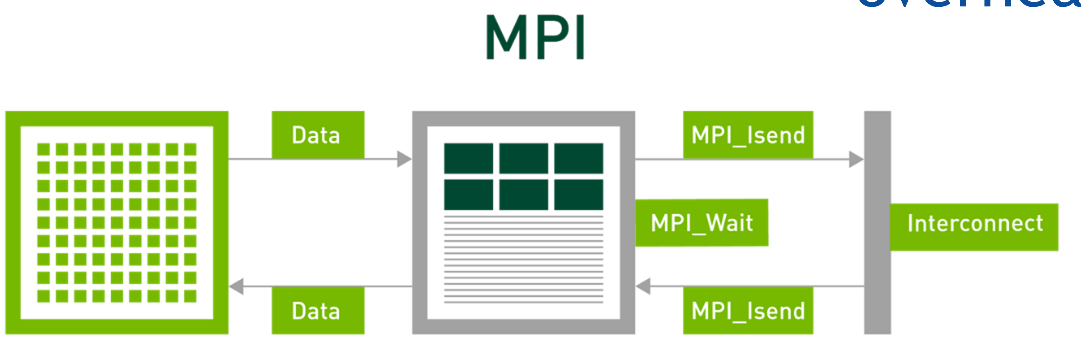
NVSHMEM

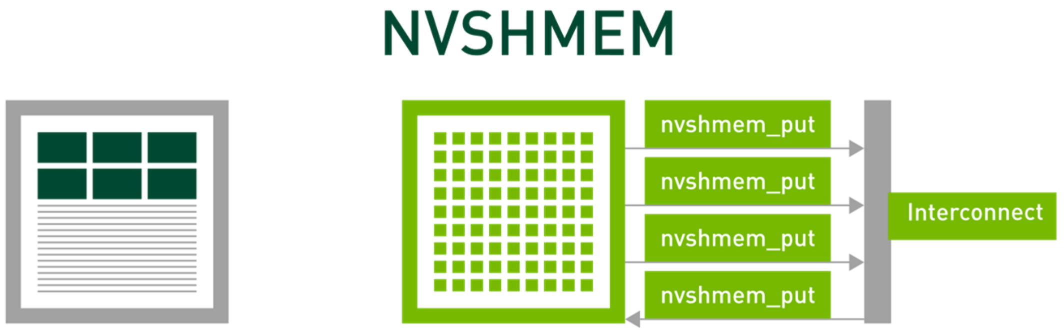
NVSHMEM

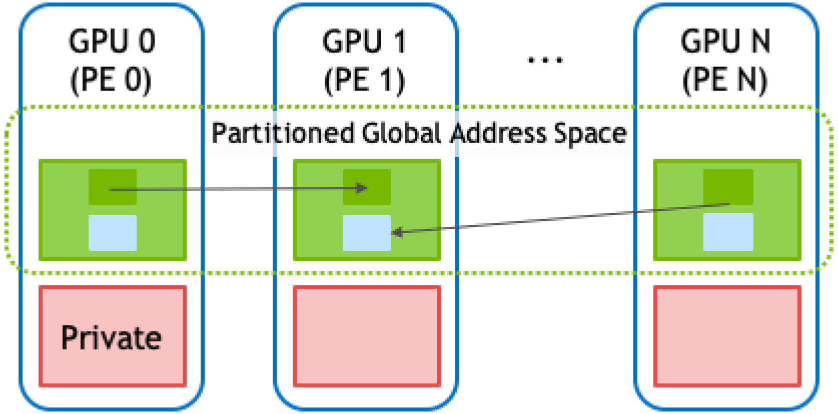
- Offers **GPU-centric** control of communication from within the parallel region
- For global addressing to local, peer, remote, offers a unified **abstraction**

MULTI-NODE MULTI-GPU: NVSHMEM

GPU-initiated communication to reduce CPU synchronization overheads







Memory model

Left Shift device kernel: NVSHMEM's Hello World

```
__global__ void simple_shift(int *destination) {
    int mype = nvshmem_my_pe();
    int npes = nvshmem_n_pes();
    int peer = (mype + 1) % npes;

    nvshmem_int_p(destination, mype, peer);
}
```

NVSHMEM

Programming abstraction for comms among GPUs

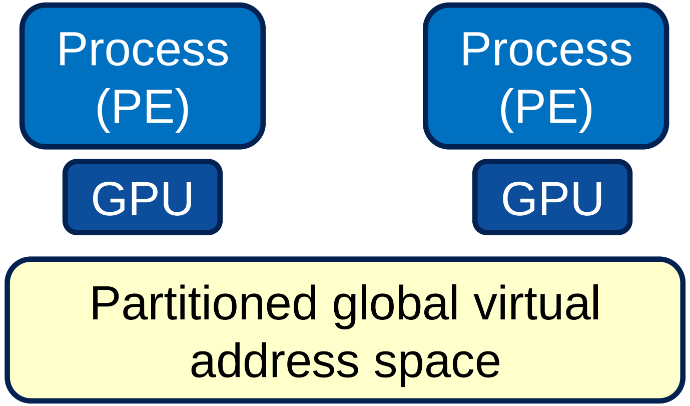
shmem_init – set up communication, including CUDA P2P, IPC

shmem_malloc – get pointer to perform allocation on local GPU

shmem_ptr – given a PE, get the base address of the symmetric heap on remote GPU

Inside or outside of GPU kernel

- ld/st using offsets relative to that base address
- put/get for abstracted communications
- collectives



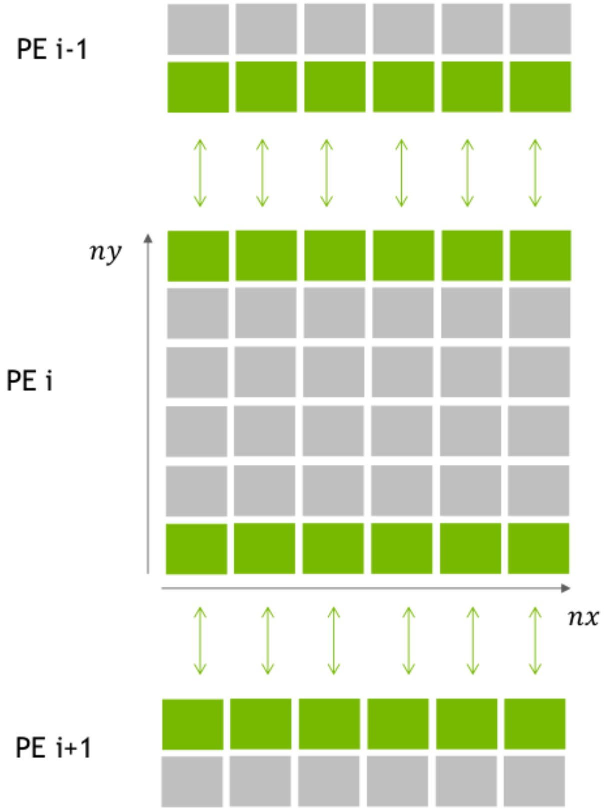
MULTI-NODE MULTI-GPU: NVSHMEM

```
// Using NVSHMEM with MPI
nvshmemx_init_attr_t attr;
MPI_Comm comm = MPI_COMM_WORLD;
attr.mpi_comm = &comm;

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

 cudaGetDeviceCount(&ndevices);
cudaSetDevice(rank % ndevices);
nvshmemx_init_attr(NVSHMEMX_INIT_WITH_MPI_COMM, &attr);

// Heap allocation
void *nvshmem_malloc(size_t size)
```

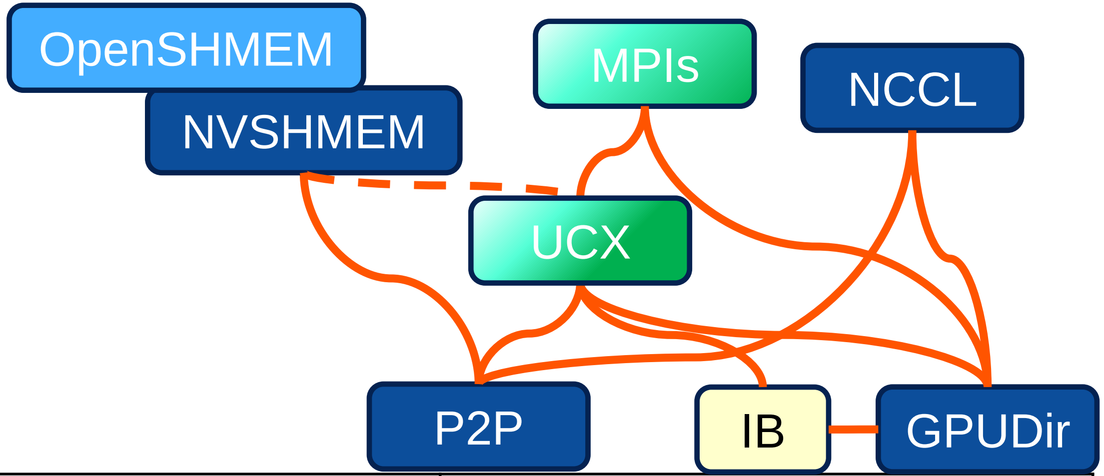
Basic APIs

```
__global__ void stencil_single_step(float *u, float *v, …) {
    int ix = get_ix(blockIdx, blockDim, threadIdx);
    int iy = get_iy(blockIdx, blockDim, threadIdx);
    compute(u, v, ix, iy);
    if (iy == 1)
        nvshmem_float_p(u+(ny+1)*nx+ix, u[nx+ix], top_pe);
    if (iy == ny)
        nvshmem_float_p(u+ix, u[ny*nx+ix], bottom_pe);
}
```

Thread-level communication API



WHAT TO USE WHEN



Usage	Interface	Strengths	Comments
Inter node	MPI	Coarse-grained buffers, CPU driven	Focused on OpenMPI, enabled by UCX
	GPUDirect RDMA, async	Avoid staging, more control at GPU	Implemented in UCX, network and storage drivers
Intra node	GPUDirect P2P	Avoid staging	Enables multiple GPUs/node
	NVSHMEM	Symmetric allocation, ease of use wrt CUDA, less overhead than MPI	Enables direct load/store over NVLink
Both	NCCL	Optimized collectives	Throughput and latency

Acknowledgment

Learn more at

WWW.OPENHACKATHONS.ORG