ADVANCED ON-NODE GPU COMMUNICATION

Steve Abbott, OLCF User Conference Call, February 2018

AGENDA

What is GPU Direct?
Intra-process Peer-to-peer
Inter-process Communication with CUDA IPC

NVIDIA.

SUMMIT NODE
OVERVIEW

SUMMIT NODE

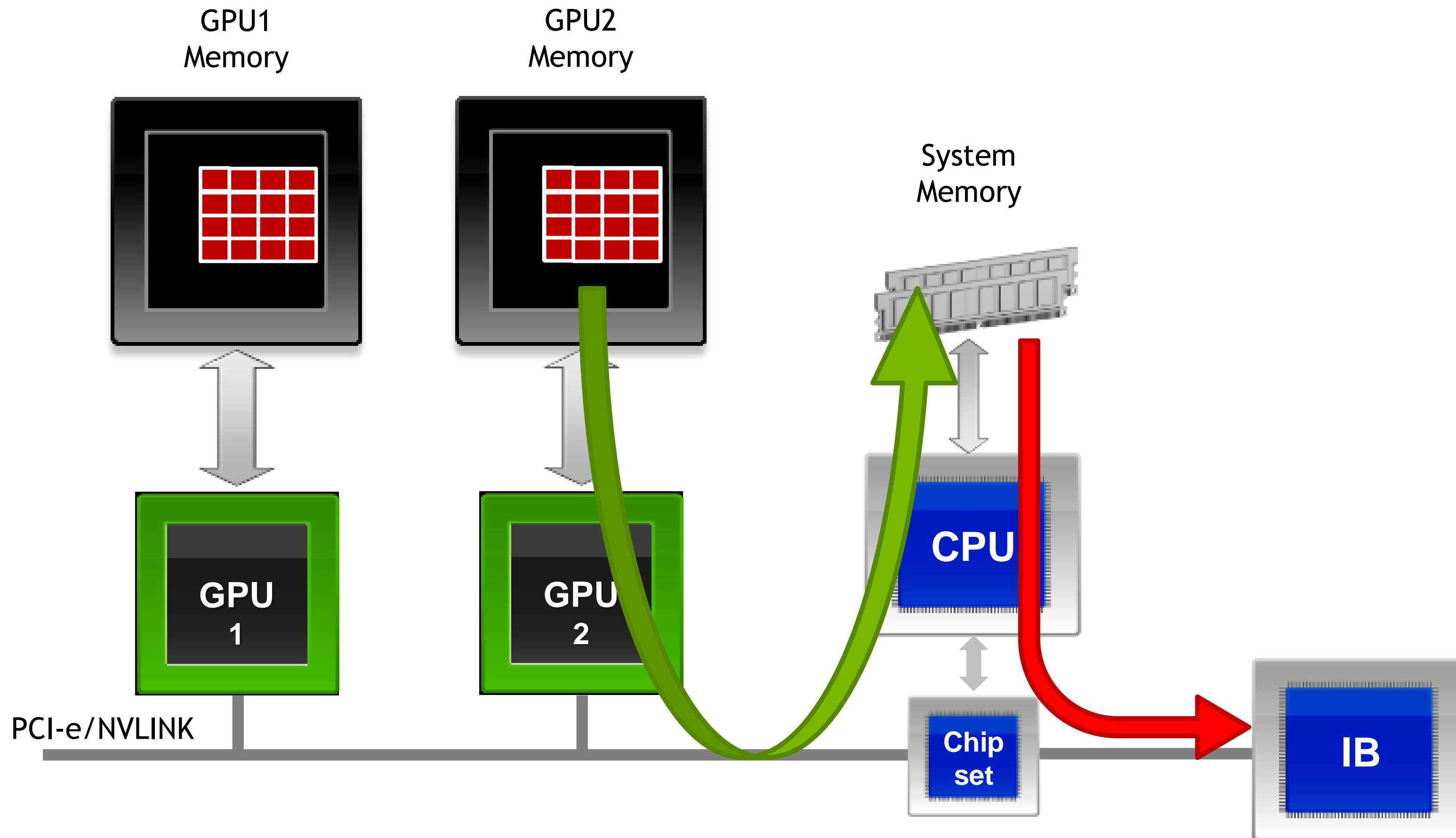
(2) IBM POWER9 + (6) NVIDIA VOLTA V100

NVIDIA

GPUDIRECT

NVIDIA GPUDIRECT™

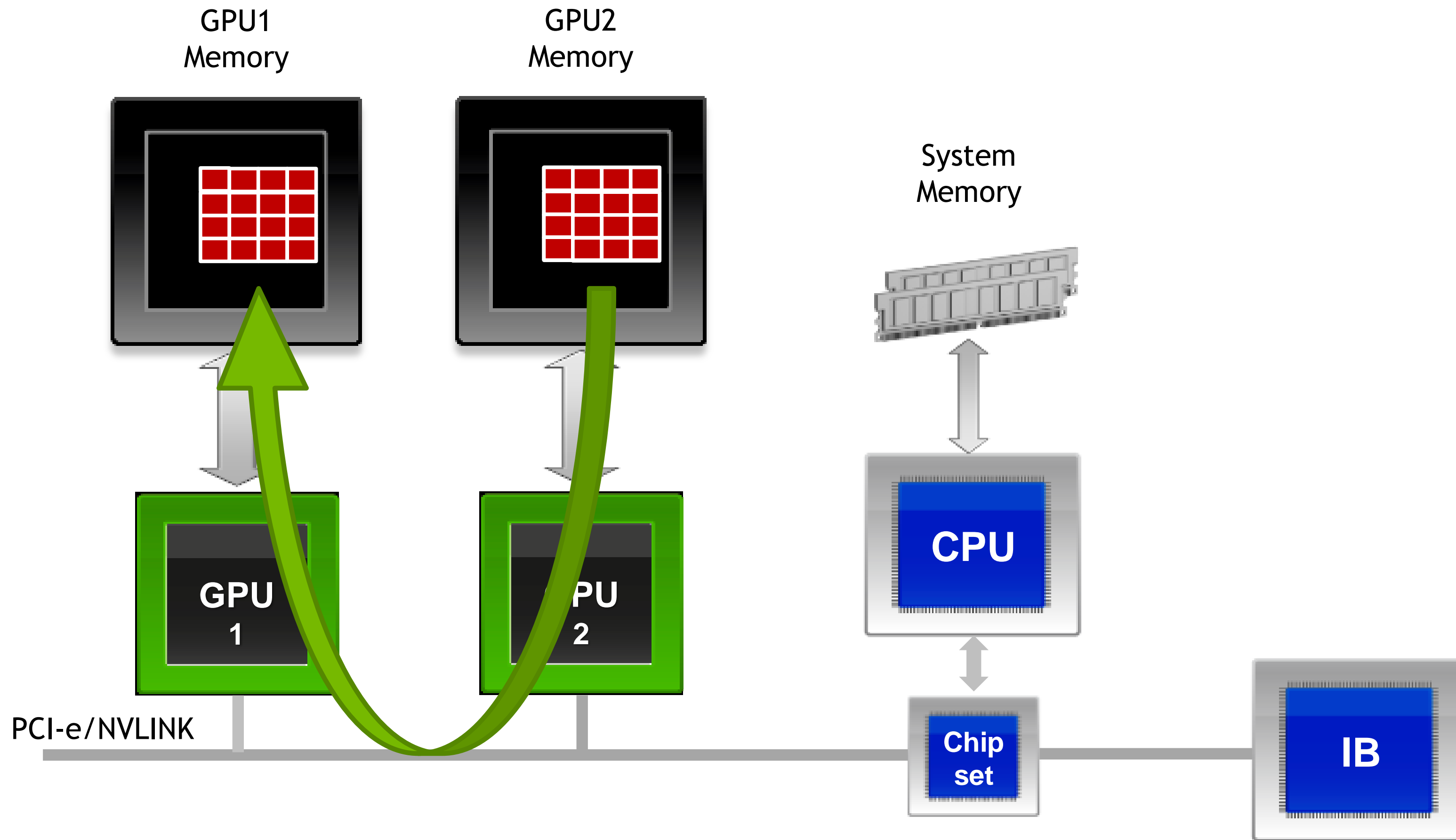
Accelerated Communication with Network & Storage Devices



NVIDIA.

NVIDIA GPUDIRECT™

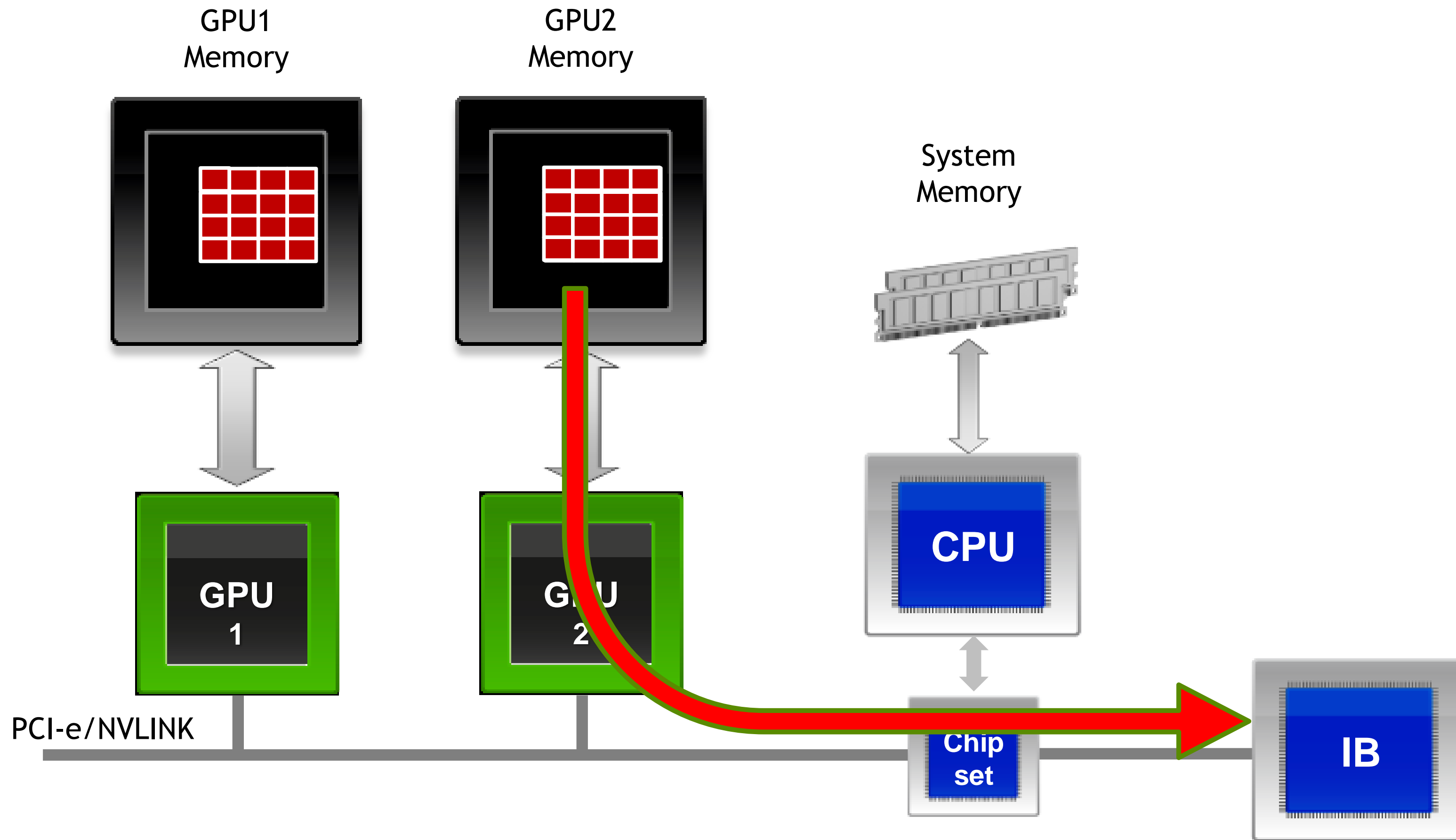
Peer to Peer Transfers



NVIDIA

NVIDIA GPUDIRECT™

Support for RDMA



NVIDIA

INTRA-PROCESS
PEER-TO-PEER

SINGLE THREADED MULTI GPU PROGRAMMING

```
while ( l2_norm > tol && iter < iter_max ) {
    for ( int dev_id = 0; dev_id < num_devices; ++dev_id ) {
        const int top = dev_id > 0 ? dev_id - 1 : (num_devices-1); const int bottom = (dev_id+1)%num_devices;
        cudaSetDevice( dev_id );
        cudaMemsetAsync(l2_norm_d[dev_id], 0 , sizeof(real) );
        jacobi_kernel<<<dim_grid,dim_block>>>( a_new[dev_id], a[dev_id], l2_norm_d[dev_id],
                                               iy_start[dev_id], iy_end[dev_id], nx );
        cudaMemcpyAsync( l2_norm_h[dev_id], l2_norm_d[dev_id], sizeof(real), cudaMemcpyDeviceToHost );
        cudaMemcpyAsync( a_new[top]+(iy_end[top]*nx), a_new[dev_id]+iy_start[dev_id]*nx, nx*sizeof(real), ...);
        cudaMemcpyAsync( a_new[bottom], a_new[dev_id]+(iy_end[dev_id]-1)*nx, nx*sizeof(real), ...);
    }
    l2_norm = 0.0;
    for ( int dev_id = 0; dev_id < num_devices; ++dev_id ) {
        cudaSetDevice( dev_id ); cudaDeviceSynchronize();
        l2_norm += *(l2_norm_h[dev_id]);
    }
    l2_norm = std::sqrt( l2_norm );
    for ( int dev_id = 0; dev_id < num_devices; ++dev_id ) std::swap(a_new[dev_id],a[dev_id]);
    iter++;
}
```


NVIDIA.

GPUDIRECT P2P

Enable P2P

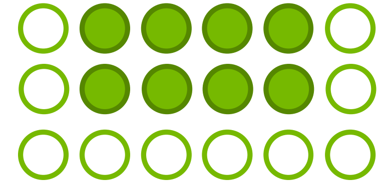
```
for ( int dev_id = 0; dev_id < num_devices; ++dev_id ) {
    cudaSetDevice( dev_id );
    const int top = dev_id > 0 ? dev_id - 1 : (num_devices-1);
    int canAccessPeer = 0;
    cudaDeviceCanAccessPeer ( &canAccessPeer, dev_id, top );
    if ( canAccessPeer )
        cudaDeviceEnablePeerAccess ( top, 0 );
    const int bottom = (dev_id+1)%num_devices;
    if ( top != bottom ) {
        cudaDeviceCanAccessPeer ( &canAccessPeer, dev_id, bottom );
        if ( canAccessPeer )
           cudaDeviceEnablePeerAccess ( bottom, 0 );
    }
}
```

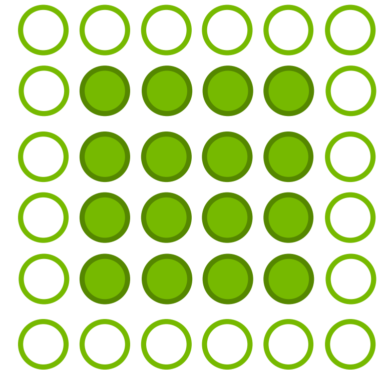

NVIDIA.

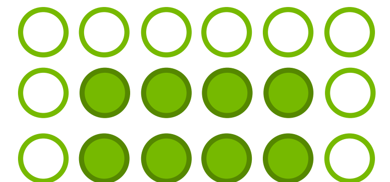
EXAMPLE JACOBI

Top/Bottom Halo

```
cudaMemcpyAsync(
  a_new[top]+(iy_end[top]*nx),
  a_new[dev_id]+iy_start[dev_id]*nx, nx*sizeof(real), ...);
```



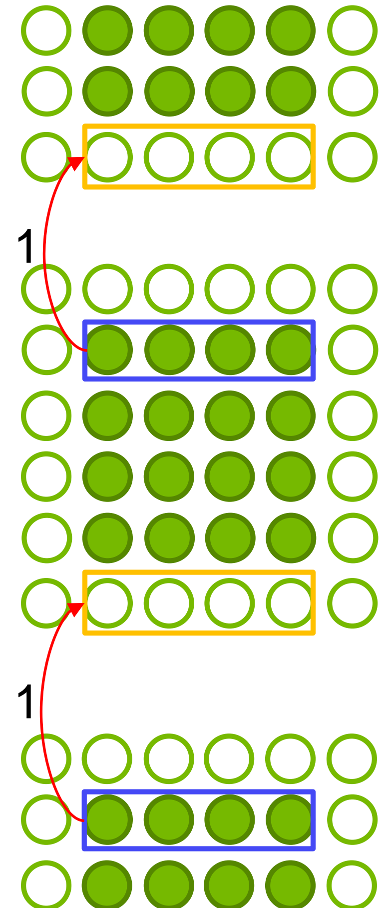




EXAMPLE JACOBI

Top/Bottom Halo

```
cudaMemcpyAsync(
  a_new[top]+(iy_end[top]*nx),
  a_new[dev_id]+iy_start[dev_id]*nx, nx*sizeof(real), ...);
```



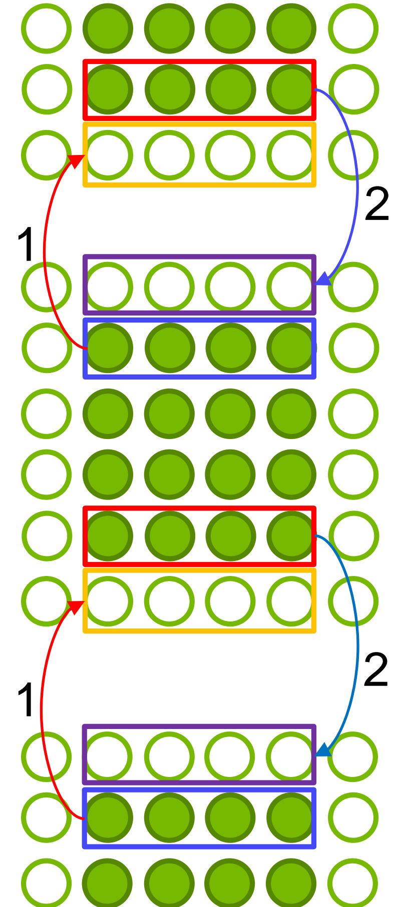
NVIDIA

EXAMPLE JACOBI

Top/Bottom Halo

```
cudaMemcpyAsync(
   a_new[top]+(iy_end[top]*nx),
   a_new[dev_id]+iy_start[dev_id]*nx, nx*sizeof(real), ...);


cudaMemcpyAsync(
   a_new[bottom],
   a_new[dev_id]+(iy_end[dev_id]-1)*nx, nx*sizeof(real), ...);
```



NVIDIA.

CUDA IPC

MULTIPLE PROCESS, SINGLE GPU W/O MPI!

```
while ( l2_norm > tol && iter < iter_max ) {
    const int top = dev_id > 0 ? dev_id - 1 : (num_devices-1); const int bottom = (dev_id+1)%num_devices;
    cudaSetDevice( dev_id );
    cudaMemsetAsync(l2_norm_d[dev_id], 0 , sizeof(real) );
    jacobi_kernel<<<dim_grid,dim_block>>>( a_new[dev_id], a[dev_id], l2_norm_d[dev_id],
                                           iy_start[dev_id], iy_end[dev_id], nx );
    cudaMemcpyAsync( l2_norm_h[dev_id], l2_norm_d[dev_id], sizeof(real), cudaMemcpyDeviceToHost );
    cudaMemcpyAsync( a_new[top]+(iy_end[top]*nx), a_new[dev_id]+iy_start[dev_id]*nx, nx*sizeof(real), ...);
    cudaMemcpyAsync( a_new[bottom], a_new[dev_id]+(iy_end[dev_id]-1)*nx, nx*sizeof(real), ...);

    l2_norm = 0.0;
    for ( int dev_id = 0; dev_id < num_devices; ++dev_id ) {
        l2_norm += *(l2_norm_h[dev_id]);
    }
    l2_norm = std::sqrt( l2_norm );
    std::swap(a_new[dev_id],a[dev_id]);
    iter++;
}
```


NVIDIA.

GPUDIRECT P2P

Enable CUDA Inter-Process Communication (IPC)!

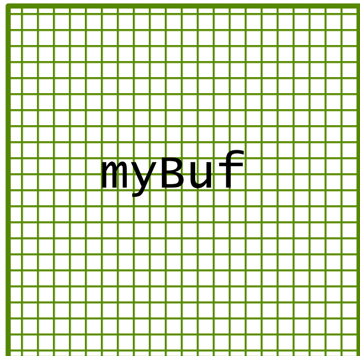
```
cudaSetDevice( dev_id );
// Allocate and fill my device buffer
cudaMalloc((void **) &myBuf, nbytes);
cudaMemcpy((void *) myBuf, (void*) buf, nbytes, cudaMemcpyHostToDevice);
// Get my IPC handle
cudaIpcMemHandle_t myIpc;
cudaIpcGetMemHandle(&myIpc, myBuf);
```


NVIDIA.

GPUDIRECT P2P

Enable CUDA Inter-Process Communication (IPC)!

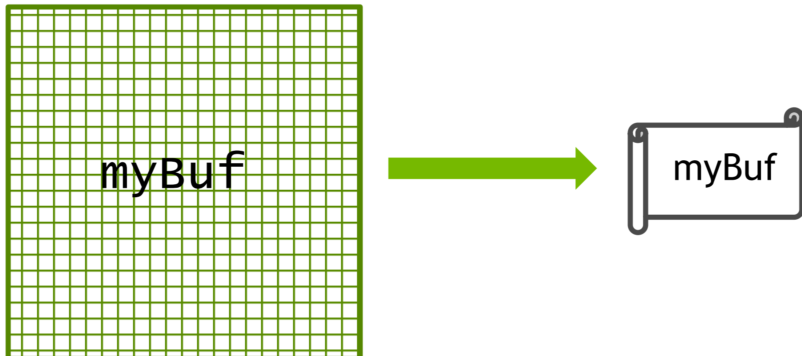
```
cudaSetDevice( dev_id );
// Allocate and fill my device buffer
cudaMalloc((void **) &myBuf, nbytes);
cudaMemcpy((void *) myBuf, (void*) buf, nbytes, cudaMemcpyHostToDevice);
// Get my IPC handle
cudaIpcMemHandle_t myIpc;
cudaIpcGetMemHandle(&myIpc, myBuf);
```



GPUDIRECT P2P

Enable CUDA Inter-Process Communication (IPC)!

```
cudaSetDevice( dev_id );
// Allocate and fill my device buffer
cudaMalloc((void **) &myBuf, nbytes);
cudaMemcpy((void *) myBuf, (void*) buf, nbytes, cudaMemcpyHostToDevice);
// Get my IPC handle
cudaIpcMemHandle_t myIpc;
cudaIpcGetMemHandle(&myIpc, myBuf);
```

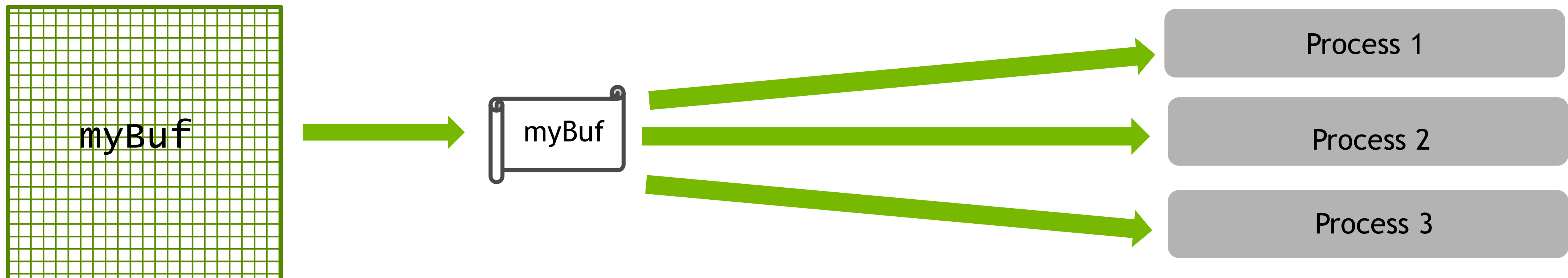


NVIDIA.

GPUDIRECT P2P

Enable CUDA Inter-Process Communication (IPC)!

```
cudaSetDevice( dev_id );
// Allocate and fill my device buffer
cudaMalloc((void **) &myBuf, nbytes);
cudaMemcpy((void *) myBuf, (void*) buf, nbytes, cudaMemcpyHostToDevice);
// Get my IPC handle
cudaIpcMemHandle_t myIpc;
cudaIpcGetMemHandle(&myIpc, myBuf);
```

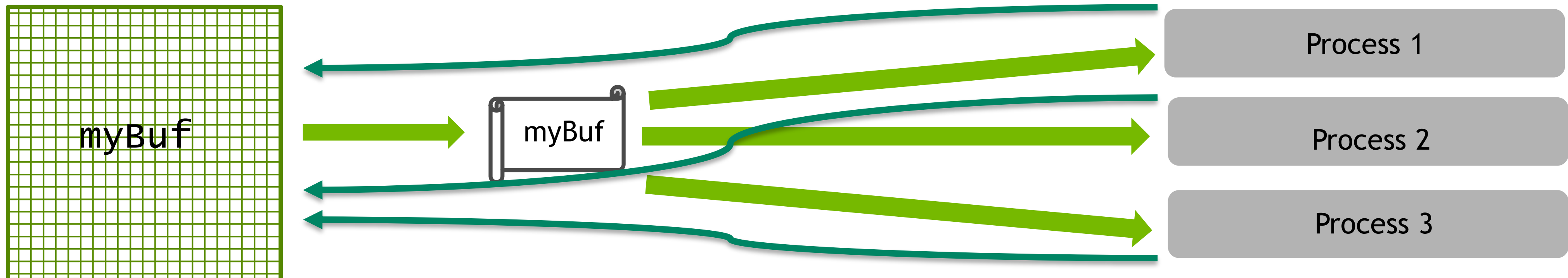


NVIDIA.

GPUDIRECT P2P

Enable CUDA Inter-Process Communication (IPC)!

```
cudaSetDevice( dev_id );
// Allocate and fill my device buffer
cudaMalloc((void **) &myBuf, nbytes);
cudaMemcpy((void *) myBuf, (void*) buf, nbytes, cudaMemcpyHostToDevice);
// Get my IPC handle
cudaIpcMemHandle_t myIpc;
cudaIpcGetMemHandle(&myIpc, myBuf);
```



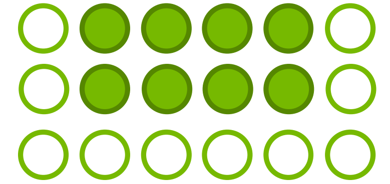
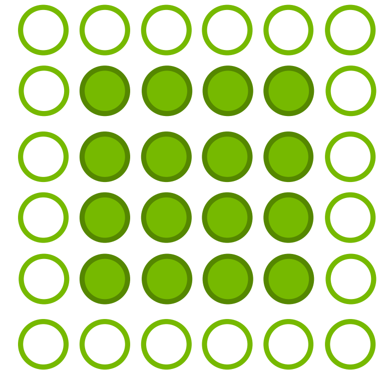
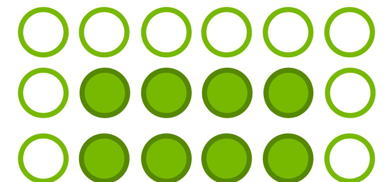
NVIDIA.

EXAMPLE JACOBI

Top/Bottom Halo

```
// Open their Ipc Handle onto a pointer
cudaIpcOpenMemHandle((void **) &a_new[top], topIpc,
    cudaIpcMemLazyEnablePeerAccess); cudaCheckError();

cudaMemcpyAsync(
  a_new[top]+(iy_end[top]*nx),
  a_new[dev_id]+iy_start[dev_id]*nx, nx*sizeof(real), ...);
```

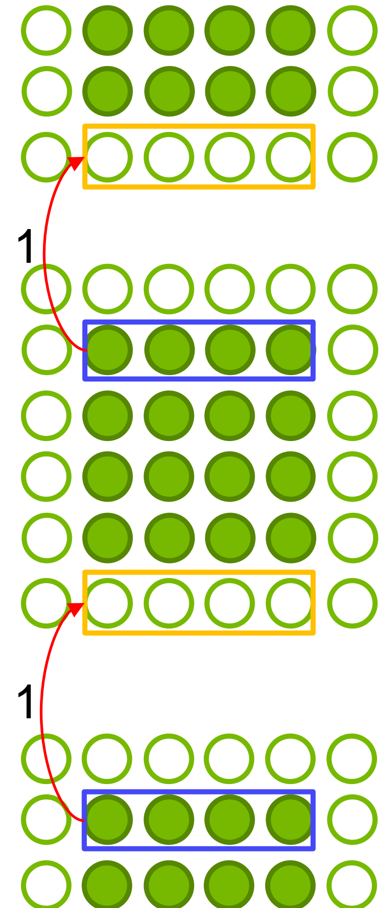
NVIDIA.

EXAMPLE JACOBI

Top/Bottom Halo

```
cudaIpcOpenMemHandle((void **) &a_new[top], topIpc,
    cudaIpcMemLazyEnablePeerAccess); cudaCheckError();

cudaMemcpyAsync(
  a_new[top]+(iy_end[top]*nx),
  a_new[dev_id]+iy_start[dev_id]*nx, nx*sizeof(real), ...);
```



NVIDIA.

EXAMPLE JACOBI

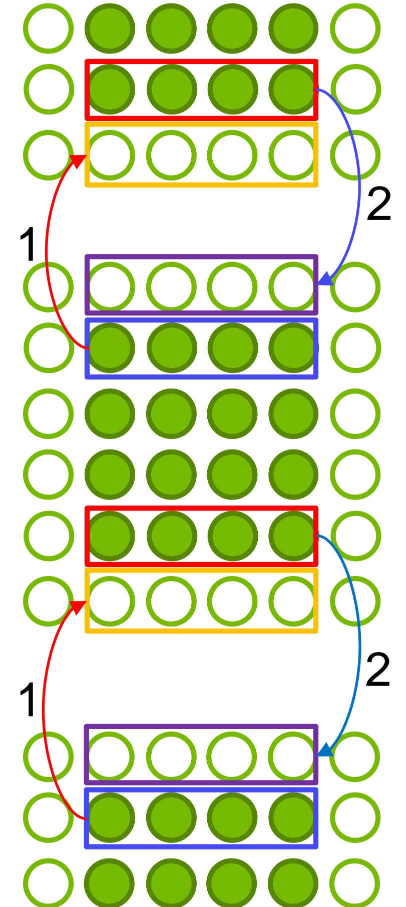
Top/Bottom Halo

```
cudaIpcOpenMemHandle((void **) &a_new[top], topIpc,
     cudaIpcMemLazyEnablePeerAccess); cudaCheckError();

cudaMemcpyAsync(
  a_new[top]+(iy_end[top]*nx),
  a_new[dev_id]+iy_start[dev_id]*nx, nx*sizeof(real), ...);


cudaIpcOpenMemHandle((void **) &a_new[bottom], bottomIpc,
     cudaIpcMemLazyEnablePeerAccess); cudaCheckError();

cudaMemcpyAsync(
  a_new[bottom],
  a_new[dev_id]+(iy_end[dev_id]-1)*nx, nx*sizeof(real), ...);
```



PROFILING NVLINK USAGE

PROFILING NVLINK USAGE

Using nvprof+NVVP

Run nvprof multiple times to collect metrics

```
jsrun <args> nvprof --output-profile profile.<metric>.%q{OMPI_COMM_WORLD_RANK}\
                     --aggregate-mode off --event-collection-mode continuous \
                      --metrics <metric> -f
```

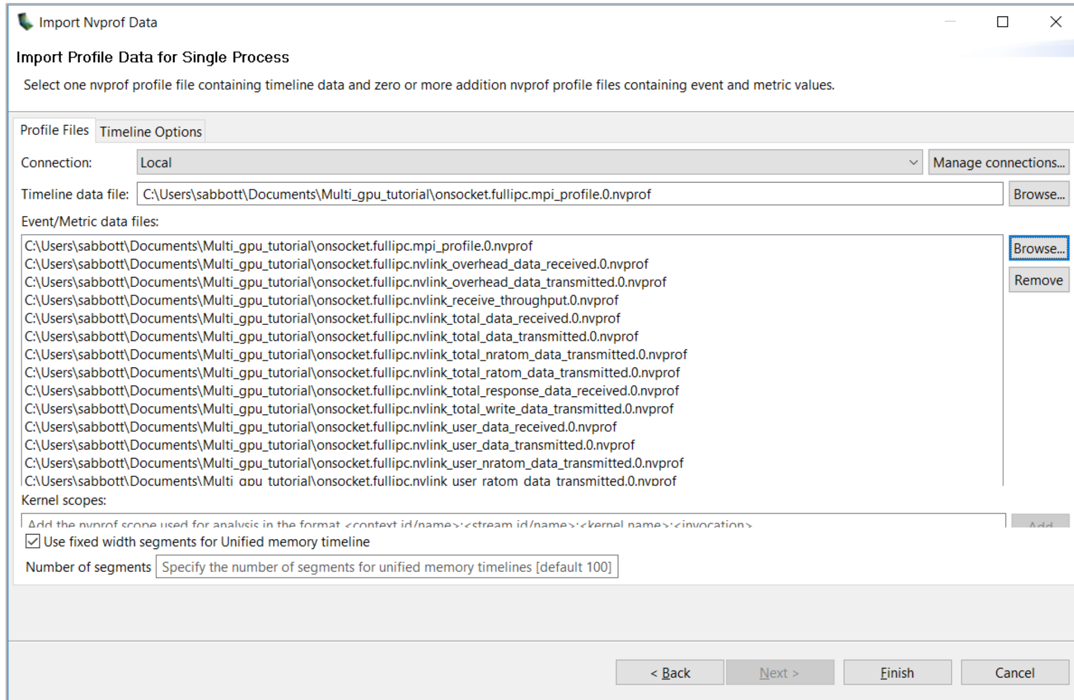
Use `--query-metrics` and `--query-events` for full list of metrics (-m) or events (-e)

Combine with an MPI annotated timeline file for full picture

NVIDIA.

PROFILING NVLINK USAGE

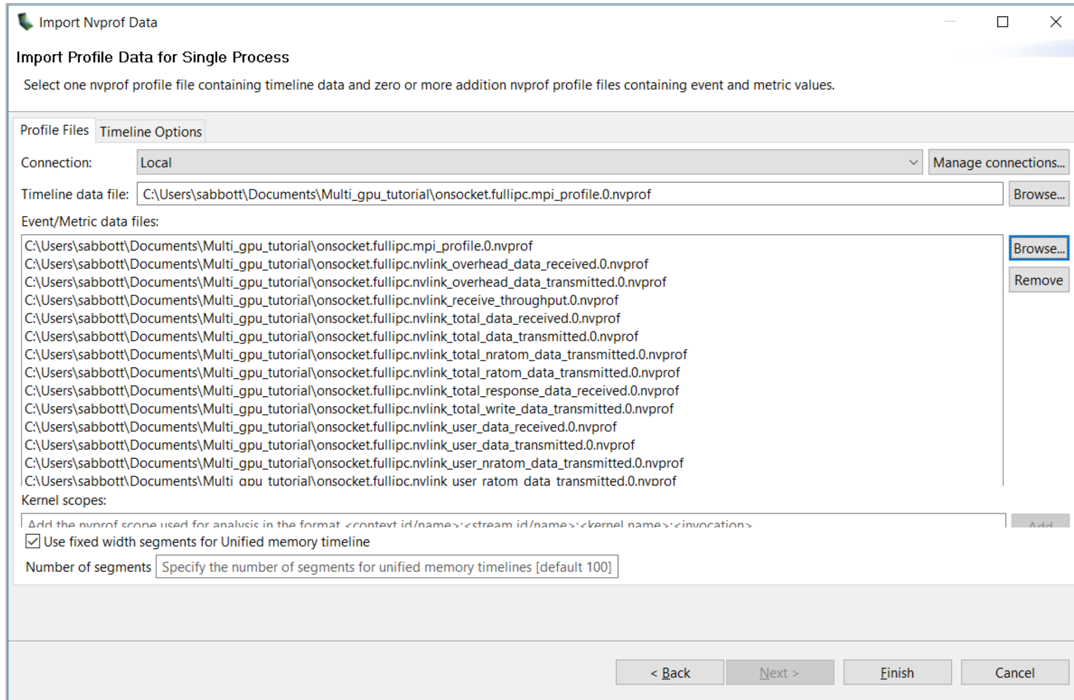
Using nvprof+NVVP

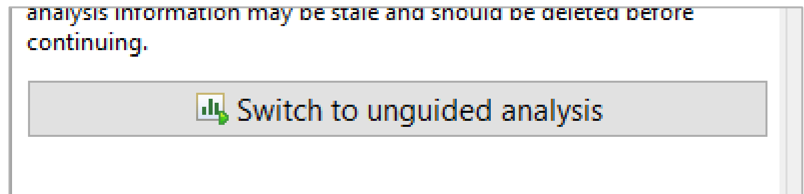


NVIDIA.

PROFILING NVLINK USAGE

Using nvprof+NVVP

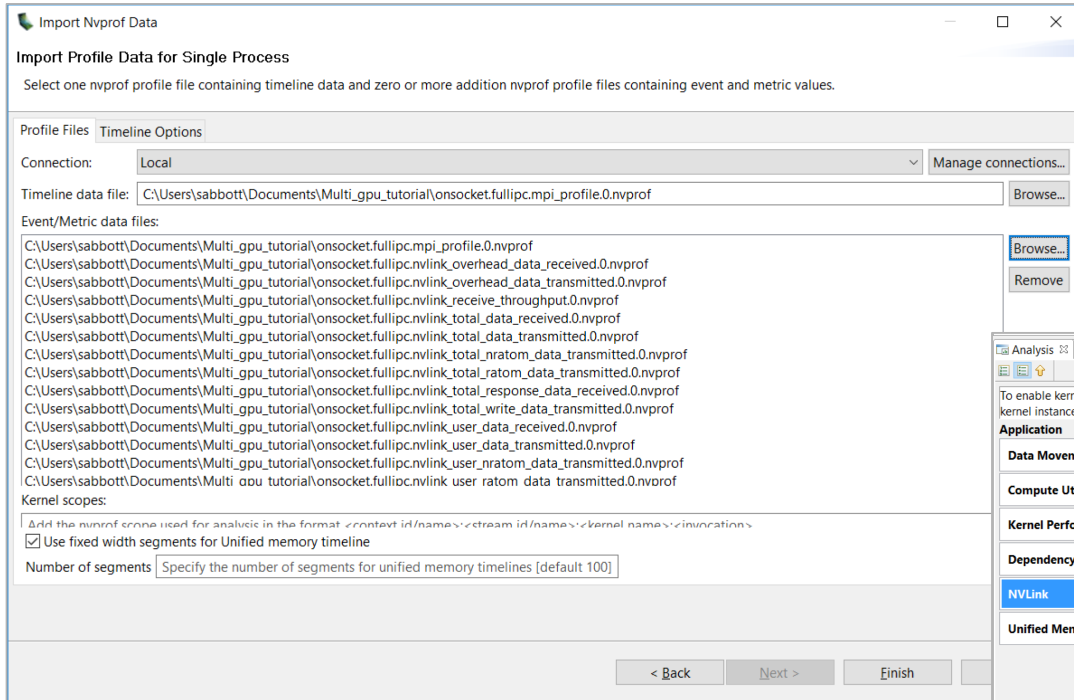




NVIDIA

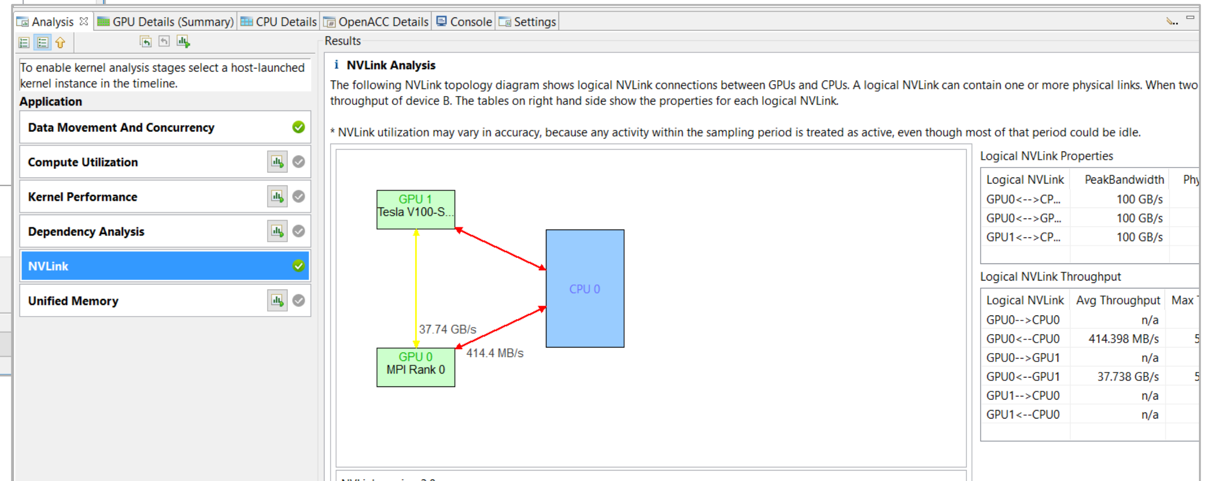
PROFILING NVLINK USAGE

Using nvprof+NVVP



analysis information may be stale and should be deleted before continuing.

Switch to unguided analysis



NVIDIA

SUMMARY

GPU TO GPU COMMUNICATION

- CUDA aware MPI functionally portable
  - OpenACC/MP interoperable
  - Performance may vary between on/off node, socket, HW support for GPU Direct
  - Unified memory support varies between implementations, but it becoming common
- Single-process, multi-GPU
  - Enable peer access for straight forward on-node transfers
- Multi-process, single-gpu
  - Pass CUDA IPC handles for on-node copies
- Combine for more flexibility/complexity!

NVIDIA.

NVIDIA