MPI + X programming

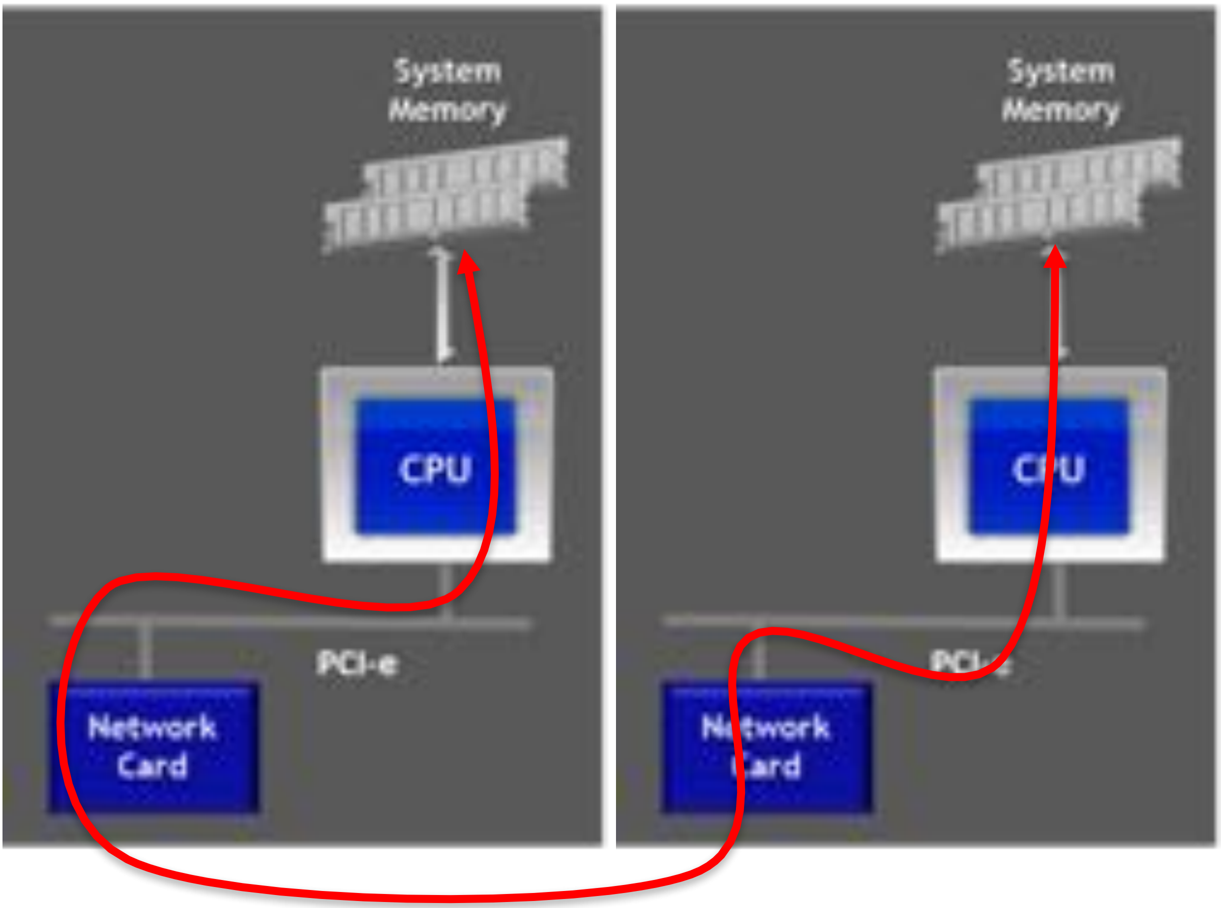
George Bosilca

CS462 – Fall 2016

https://newton.utk.edu/doc/Documentation/

Rho Cluster with GPGPU
https://newton.utk.edu/doc/Documentation/Systems/RhoCluster

MPI



- Each programming paradigm only covers a particular spectrum of the hardware capabilities
  - MPI is about moving data between distributed memory machines
  - CUDA is about accessing the sheer computations power of a single GPU
  - OpenMP is about taking advantage of the multicores architectures
- What is involved in moving data between 2 machines
  - Bus (PCI/PCI-X)
  - Memory (pageable, pinned, virtual)
  - OS (security)

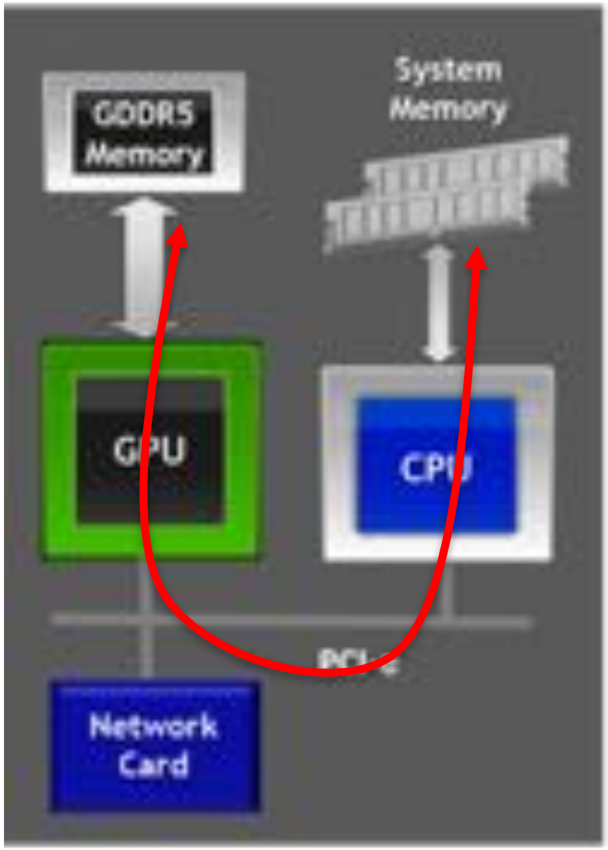
Applications need to fully take advantage of all available hardware capabilities . It became imperative to combine different programming paradigms together !

PCI-X performance

PCI Express link performance[27][30]

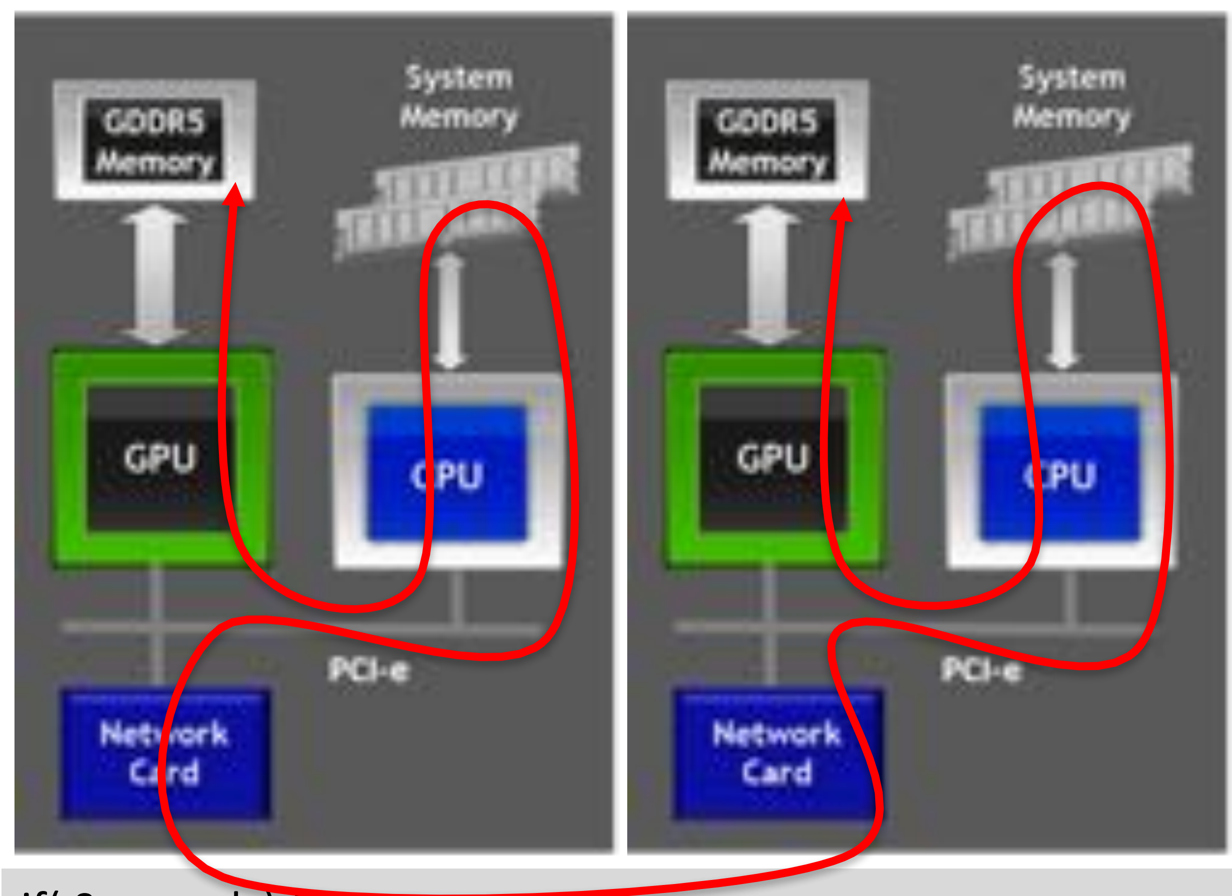
PCI Express version	Line code	Transfer rate[i]	Throughput[i]			
			x1	x4	x8	x16
1.0	8b/10b	2.5 GT/s	250 MB/s	1 GB/s	2 GB/s	4 GB/s
2.0	8b/10b	5 GT/s	500 MB/s	2 GB/s	4 GB/s	8 GB/s
3.0	128b/130b	8 GT/s	984.6 MB/s	3.938 GB/s	7.877 GB/s	15.754 GB/s
4.0 (*expected in 2017*)	128b/130b	16 GT/s	1.969 GB/s	7.877 GB/s	15.754 GB/s	31.508 GB/s
5.0 (*far future*)[28][29]	128b/130b	32 or 25 GT/s[ii]	3.9, or 3.08 GB/s	15.8, or 12.3 GB/s	31.5, or 24.6 GB/s	63.0, or 49.2 GB/s

CUDA



- The CPU is the main driver, it launches kernels on the GPU that perform computations
  sum<<<1,1>>>(2, 3, device_z);
  - Data must be moved between main memory and GPU prior to the computations
  - And must be fetched back once the computation is completed
  - In general these are explicit operations (cudaMemcpy)

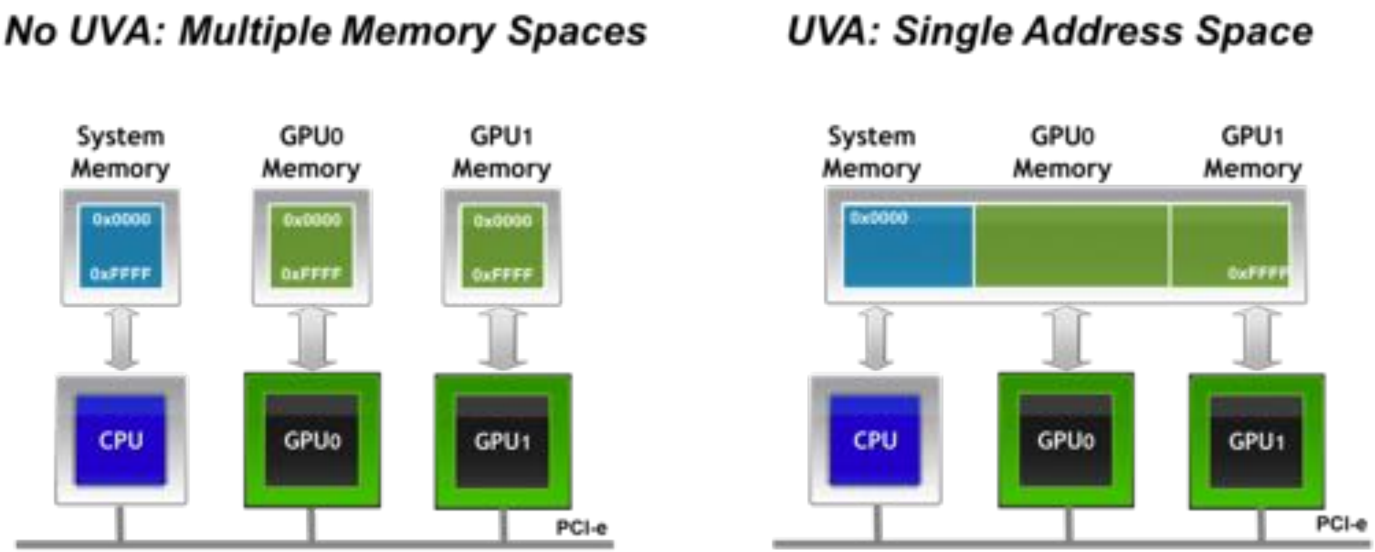
MPI + CUDA



- MPI is handling main memory while CUDA kernels update the GPU memory. Explicit memory copy from the device to the CPU is necessary to ensure coherence.

```
if( 0 == rank ) {
   cudaMemcpy(buf_host, buf_dev, size, cudaMemcpyDeviceToHost);
   MPI_Send(buf_host, size, MPI_CHAR, 1, tag, MPI_COMM_WORLD);
} else { // assume MPI rank 1
   MPI_Recv(buf_host, size, MPI_CHAR, 0, tag, MPI_COMM_WORLD, &status);
   cudaMemcpy(buf_dev, buf_host, size, cudaMemcpyHostToDevice);
}
```

Unified Virtual Addressing (UVA)

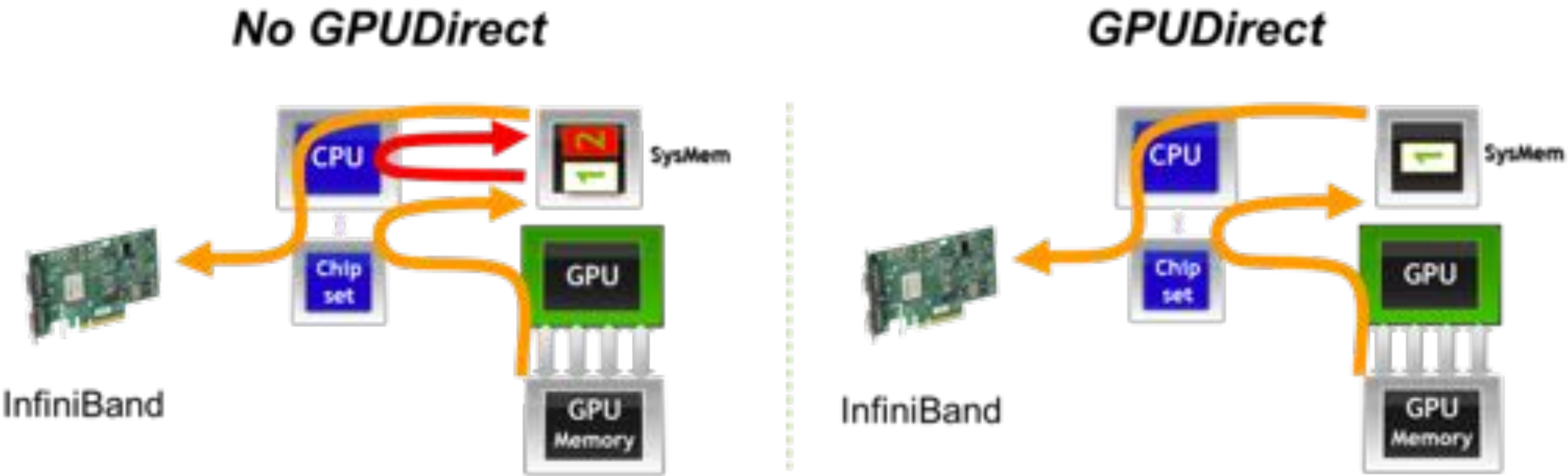


Devices have similar ranges of memory.
Impossible to know where a memory range belongs to

Devices have continuous ranges of memory (managed by the hardware and OS).
A memory address clearly identifies the hardware device hosting the memory

UVA: One address space for all CPU and GPU memory
No need to alter libraries, they can how identify on which device the memory is located

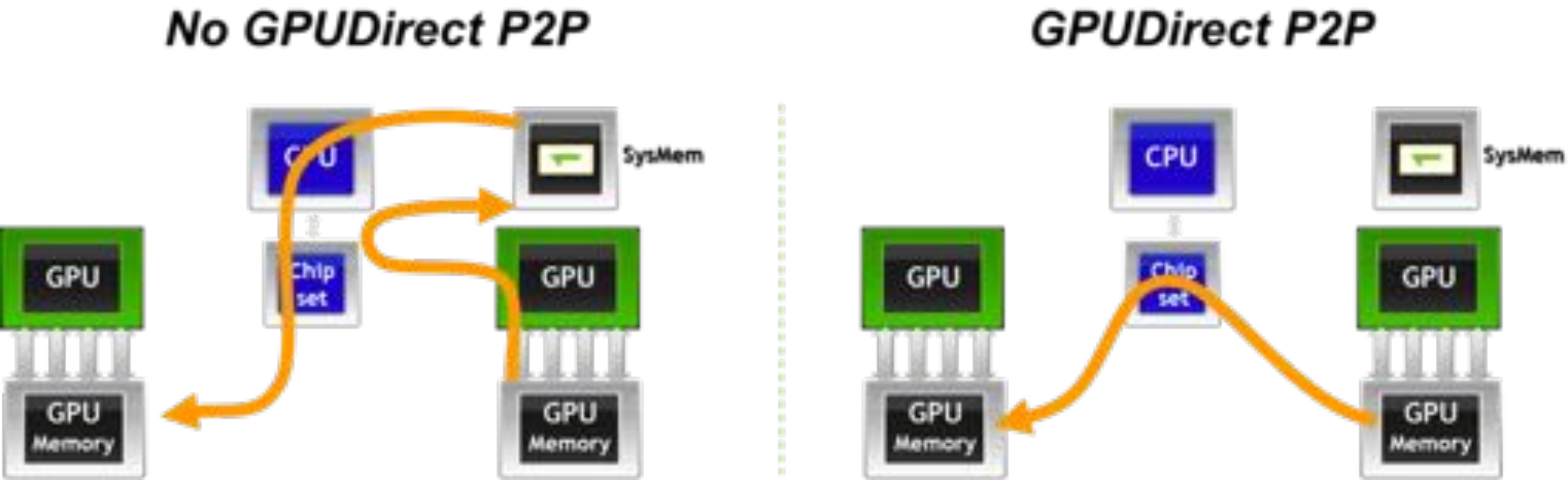
Nvidia GPUDirect



- Allowed pinned pages to be shared between different users
  - Ne need for multiple intermediary buffers to ready the data to be sent over the NiC

CUDA 3.1

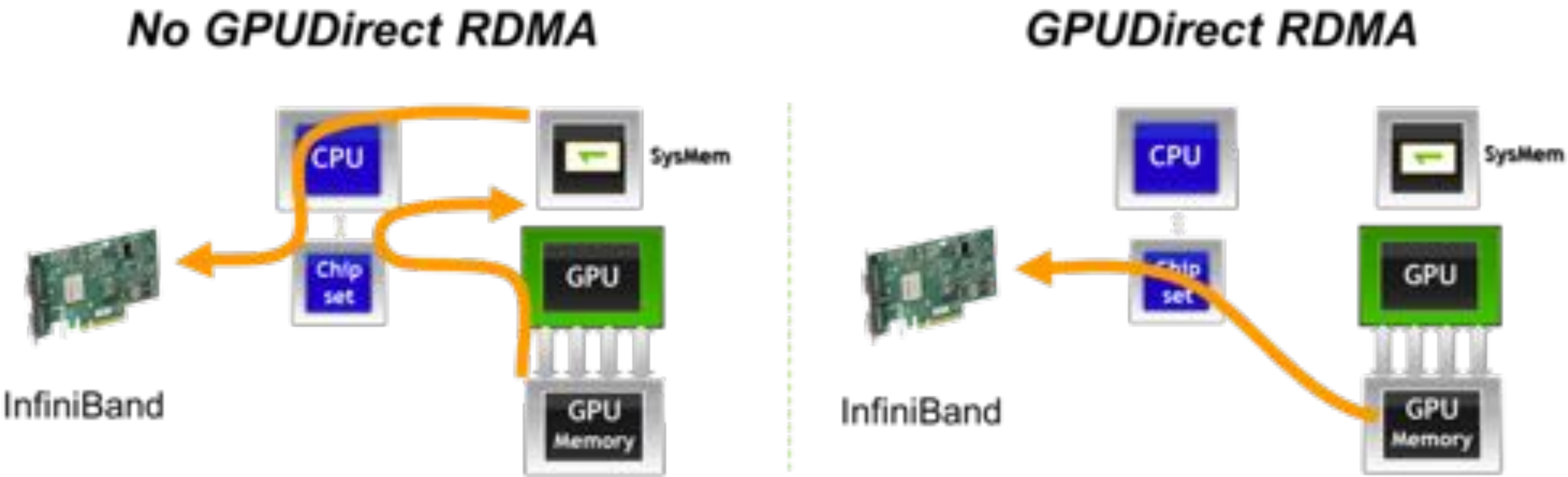
Nvidia GPUDirect P2P



- P2P (Peer-to-Peer) allows memory to be copied between devices on the same node without going through the main memory.

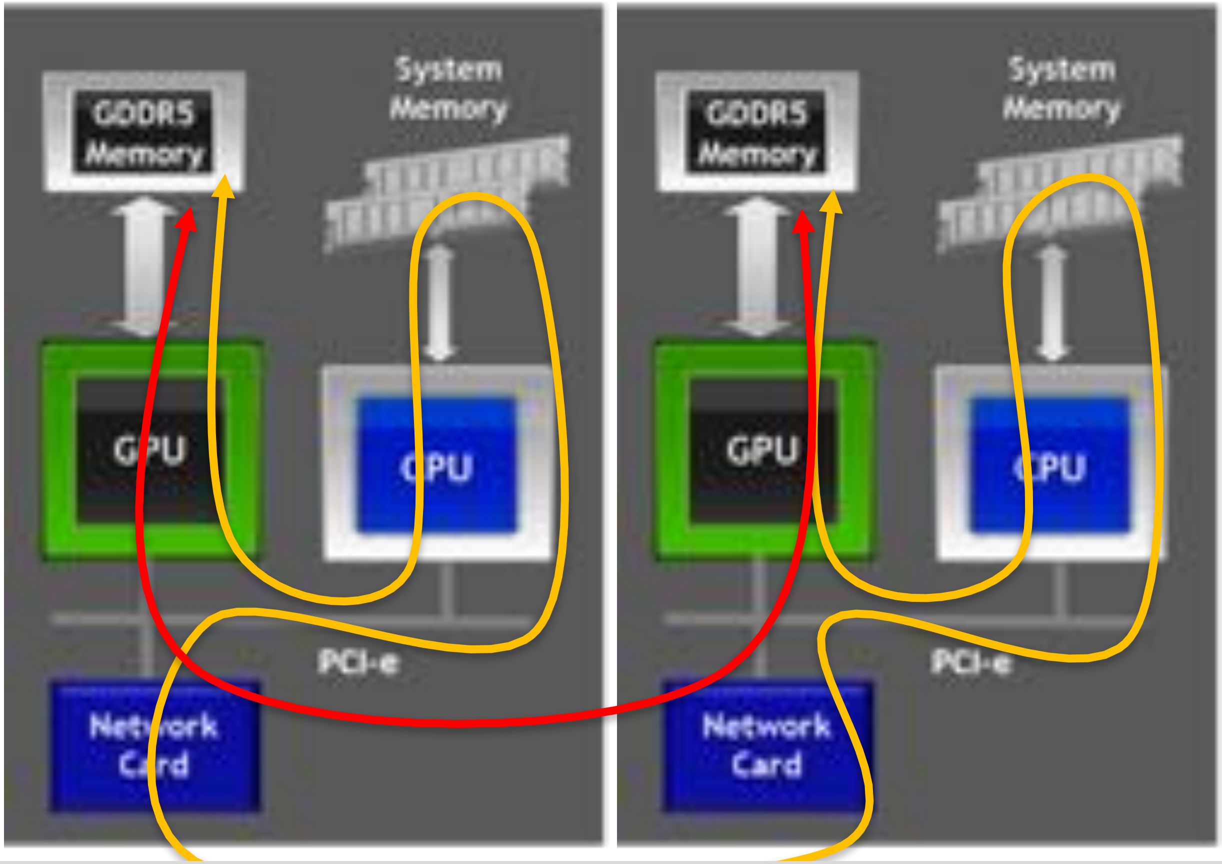
CUDA 4.0

Nvidia GPUDirect RDMA



- Push the data out of the GPU directly into the NiC (or other hardware component).
  - Implement standard parts of the PCI-X protocol

MPI + CUDA: integration/awarness

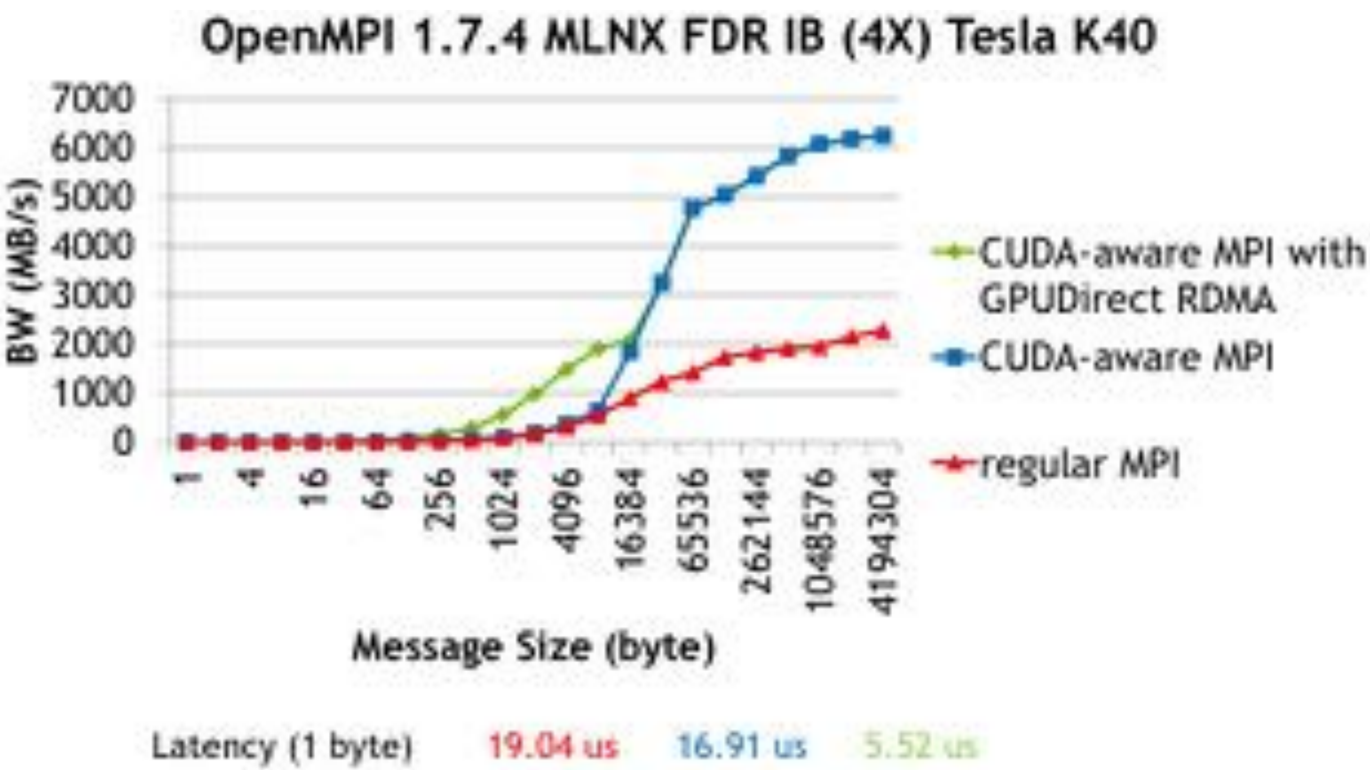


- Explicit memory copy from the device to the CPU is not necessary to ensure coherence.
- Data now flows directly between the local and remote memory (independent on the location of the memory).

```
if( 0 == rank ) {
   cudaMemcpy(buf_host, buf_dev, size, cudaMemcpyDeviceToHost);
   MPI_Send(buf_dev, size, MPI_CHAR, 1, tag, MPI_COMM_WORLD);
} else { // assume MPI rank 1
   MPI_Recv(buf_dev, size, MPI_CHAR, 0, tag, MPI_COMM_WORLD, &status);
   cudaMemcpy(buf_dev, buf_host, size, cudaMemcpyHostToDevice);
}
```

CUDA-aware MPI

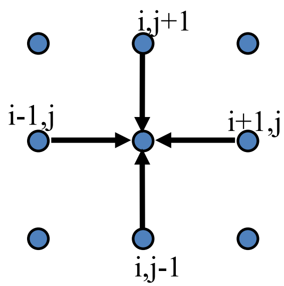
```
if( 0 == rank ) {
   MPI_Send(buf_dev, size, MPI_CHAR, 1, tag, MPI_COMM_WORLD);
} else { // assume MPI rank 1
   MPI_Recv(buf_dev, size, MPI_CHAR, 0, tag, MPI_COMM_WORLD, &status);
}
```



- MVAPICH2 1.8/1.9b
- OpenMPI 1.7 (beta)
- CRAY MPI (MPT 5.6.2)
- IBM Platform MPI (8.3)
- SGI MPI (1.08)

$$U_{i,j}^{n+1} = \frac{1}{4}\left(U_{i-1,j}^{n} + U_{i+1}^{n} + U_{i,j-1}^{n} + U_{i,j+1}^{n}\right)$$

Laplace’s equation – MPI + CUDA




```
for j = 1 to jmax
  for i = 1 to imax
    Unew(i,j) = 0.25 * ( U(i-1,j) + U(i+1,j)
                       +   U(i,j-1) + U(i,j+1))

  end for
end for
```