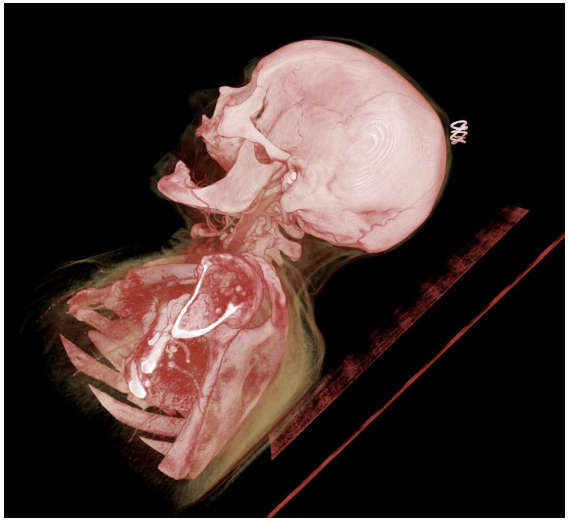
Efficient Intranode Communication in GPU-Accelerated Systems

Feng Ji, Ashwin M. Aji, James Dinan, Darius Buntinas, Pavan Balaji, Wu-chun Feng, Xiaosong Ma

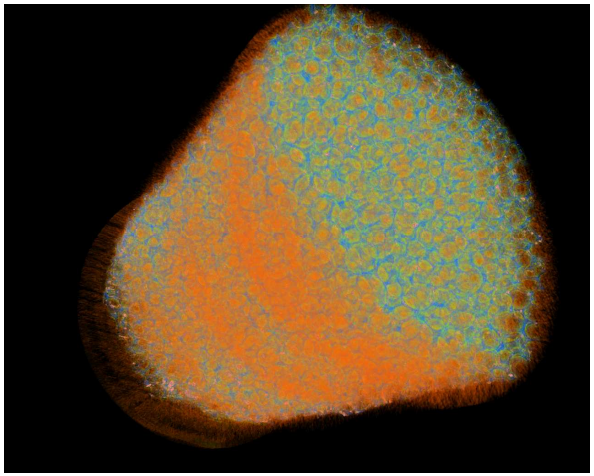
Presented by: James Dinan
James Wallace Givens Postdoctoral Fellow
Argonne National Laboratory

Fields using GPU Accelerators at Argonne

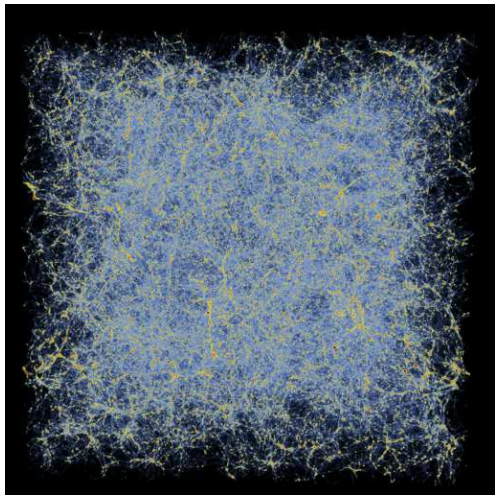
Computed Tomography



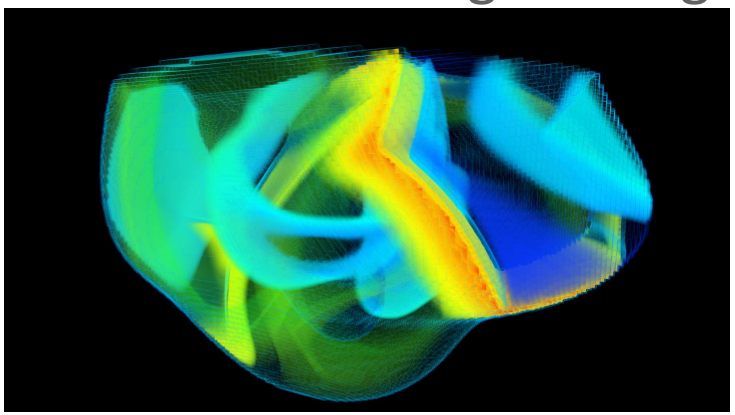
Micro-tomography



Cosmology

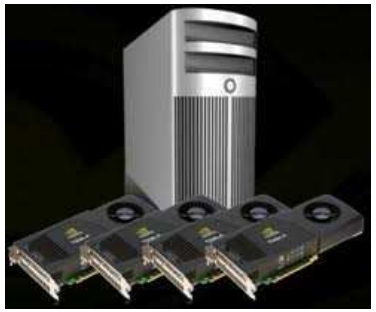


Bioengineering



Acknowledgement: Venkatram Vishwanath @ ANL

GPU-Based Supercomputers



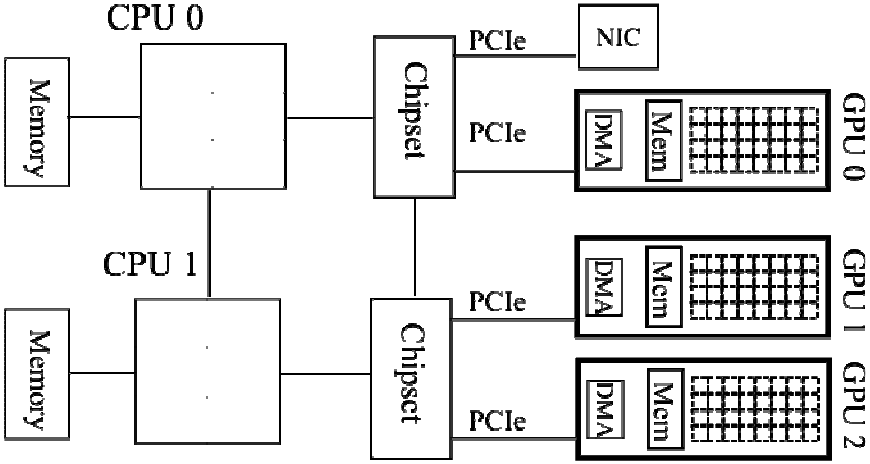
TOP500® SUPERCOMPUTER SITES

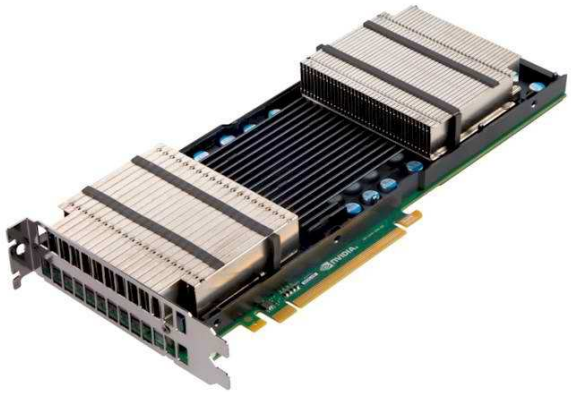
Rank	Site	Computer/Year Vendor	Cores	R_{max}	R_{peak}	Power
1	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect / 2011 Fujitsu	705024	10510.00	11280.38	12659.9
2	National Supercomputing Center in Tianjin China	NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050 / 2010 NUDT	186368	2566.00	4701.00	4040.0
3	DOE/SC/Oak Ridge National Laboratory United States	Cray XT5-HE Opteron 6-core 2.6 GHz / 2009 Cray Inc.	224162	1759.00	2331.00	6950.0
4	National Supercomputing Centre in Shenzhen (NSCS) China	Dawning TC3600 Blade System, Xeon X5650 6C 2.66GHz, Infiniband QDR, NVIDIA 2050 / 2010 Dawning	120640	1271.00	2984.30	2580.0
5	GSIC Center, Tokyo Institute of Technology Japan	HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU, Linux/Windows / 2010 NEC/HP	73278	1192.00	2287.63	1398.6

GPU-Accelerated High Performance Computing

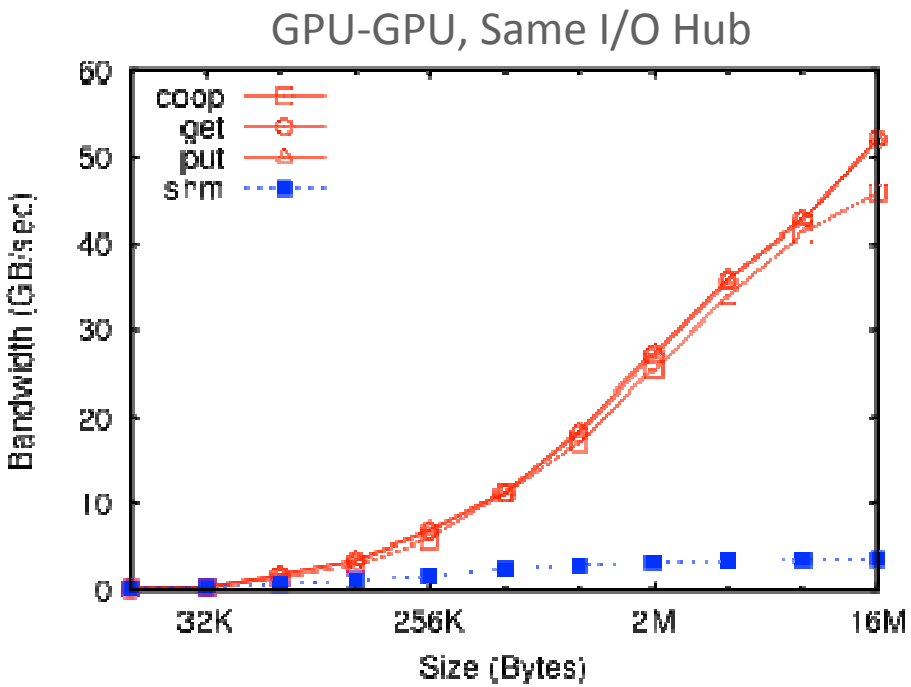
- GPUs are general purpose, highly parallel processors
  - High FLOPs/Watt and FLOPs/$
  - Unit of execution *Kernel*
  - Separate memory subsystem
  - Prog. Models: CUDA, OpenCL, …
- Clusters with GPUs are becoming common
  - Multiple GPUs per node
  - Nonuniform node architecture
  - Node topology plays role in performance
- New programmability and performance challenges for programming models and runtime systems

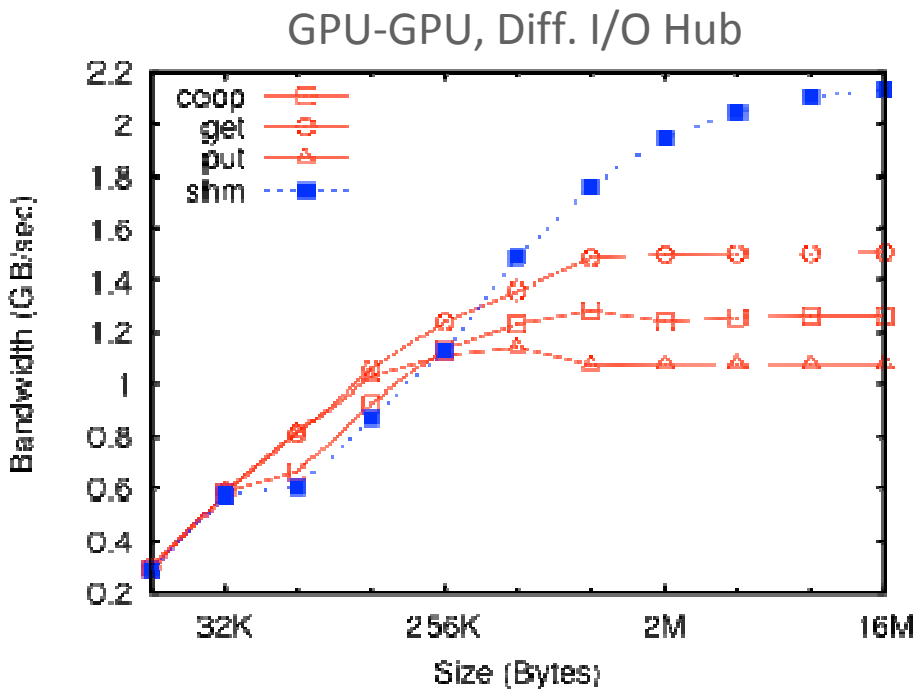
Keeneland Node Architecture





Heterogeneity and *Intra*-node GPU-GPU Xfers

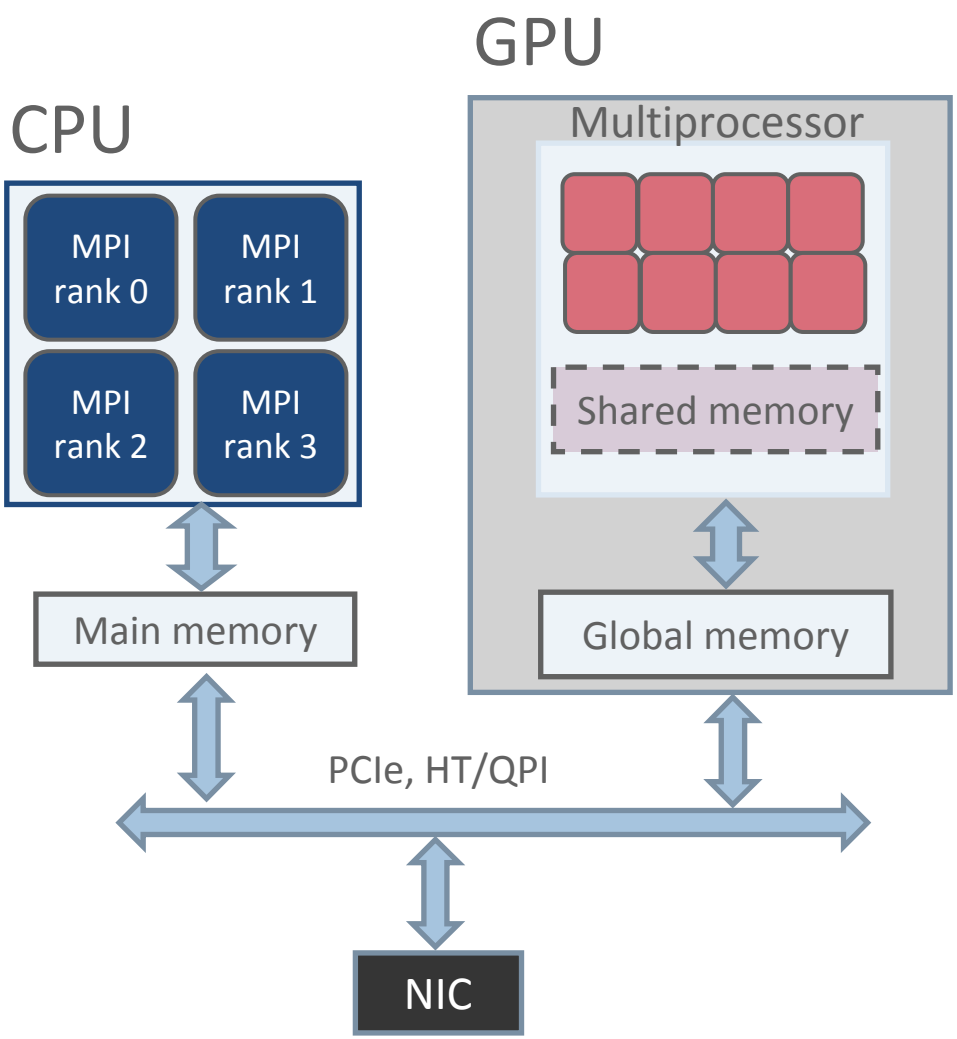




- CUDA provides GPU-GPU DMA using CUDA IPC
- Same I/O hub – DMA best
- Different I/O hubs – Shared memory best
  - Mismatch between PCIe and QPI ordering semantics

MPI-ACC: Programmability and Performance

- GPU Global memory
  - Separate address space
  - Manually managed
- Message Passing Interface
  - Most popular parallel programming model
  - Host memory only
- Integrate accelerator awareness with MPI (ANL, NCSU, VT)
  - Productivity and performance benefits



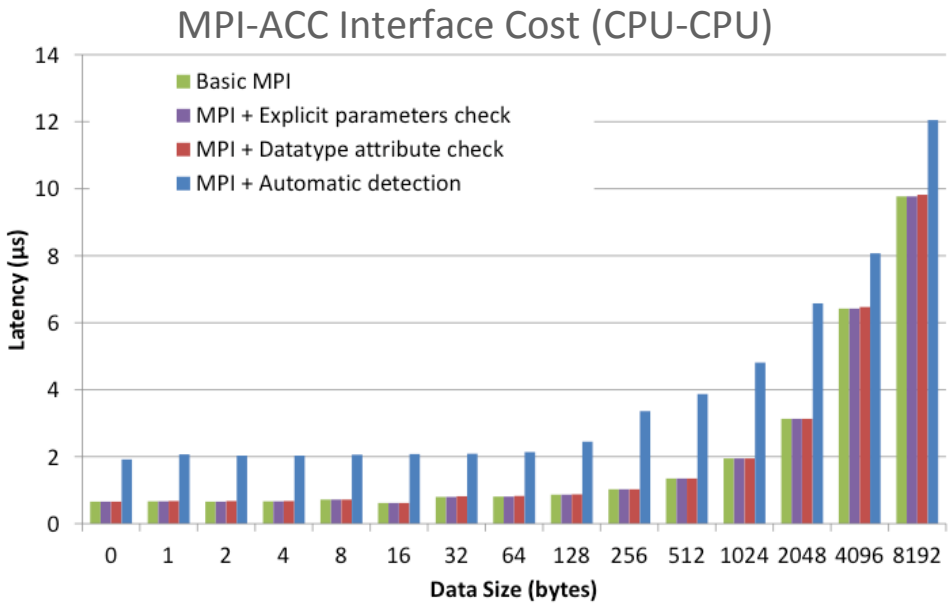
Current MPI+GPU Programming

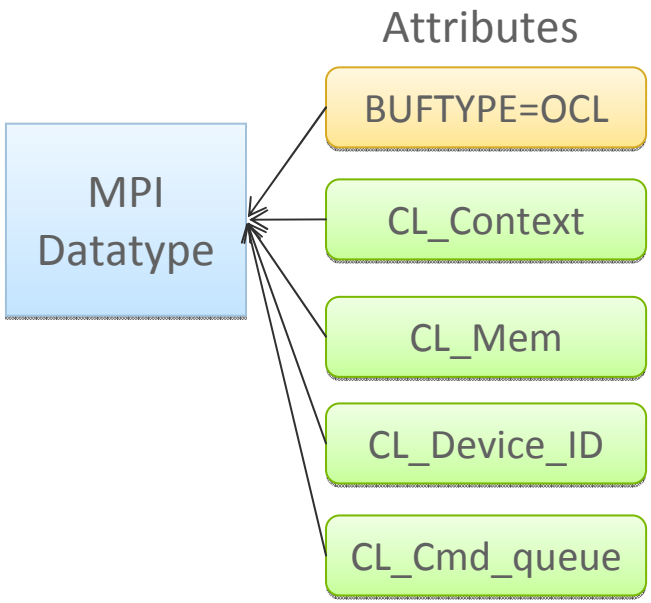
```
double *dev_buf, *host_buf;
cudaMalloc(&dev_buf, size);
cudaMallocHost(&host_buf, size);

if (my_rank == sender) { /* sender */
  computation_on_GPU(dev_buf);
  cudaMemcpy(host_buf, dev_buf, size, ...);
  MPI_Send(host_buf, size, ...);
} else {                    /* receiver */
  MPI_Recv(host_buf, size, ...);
  cudaMemcpy(dev_buf, host_buf, size, ...);
  computation_on_GPU(dev_buf);
}
```

- MPI operates on data in host memory only
- Manual copy between host and GPU memory serializes PCIe, Interconnect
  - Can do better than this, but will incur protocol overheads multiple times
- **Productivity**: Manual data movement
- **Performance**: Inefficient, unless large, non-portable investment in tuning

MPI-ACC Interface: Passing GPU Buffers to MPI



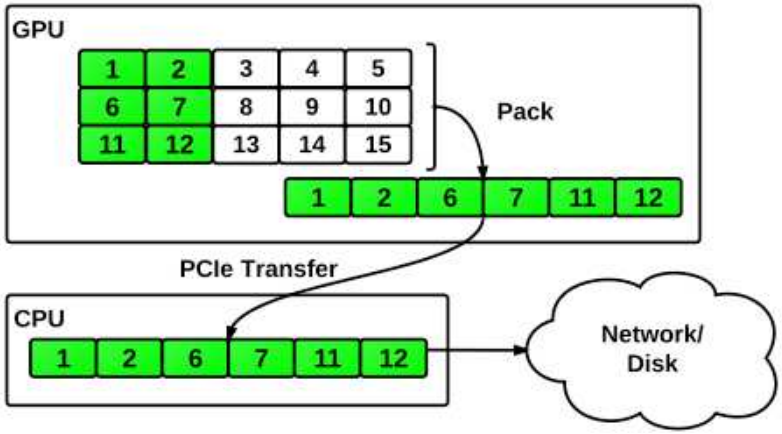


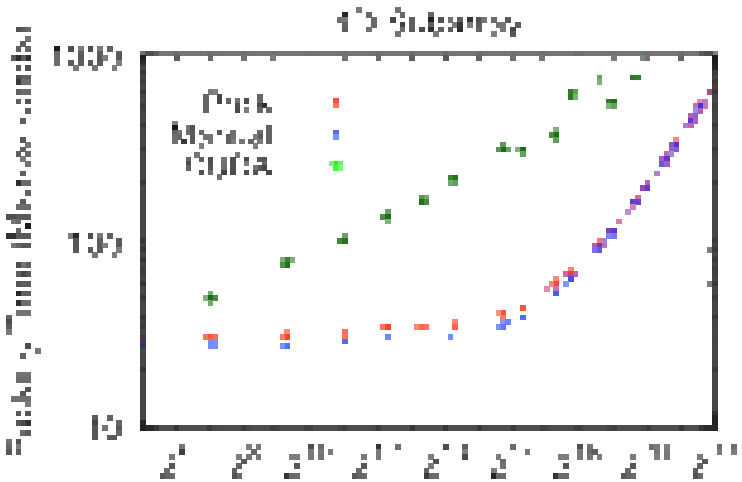
- Unified Virtual Address (UVA) space
  - Allow device pointer in MPI routines directly
  - Currently supported only by CUDA and newer NVIDIA GPUs
  - Query cost is high and added to *every* operation (CPU-CPU)
- Explicit Interface – e.g. MPI_CUDA_Send(…), overloading
- MPI Datatypes – Compatible with MPI *and* many accelerator models

MPI-ACC: Integrated, Optimized Data Movement

- Use MPI for *all* data movement
  - Support multiple accelerators and prog. models (CUDA, OpenCL, …)
  - Allow application to portably leverage system-specific optimizations
- *Inter*-node data movement [Aji HPCC'12]
  - Pipelining: Fully utilize PCIe and network links
  - GPU direct (CUDA): Multi-device pinning eliminates data copying
  - Handle caching (OpenCL): Avoid expensive command queue creation
- *Intra*-node data movement
  - Shared memory protocol [Ji ASHES'12]
    - Sender and receiver drive independent DMA transfers
  - Direct DMA protocol [Ji HPCC'12]
    - GPU-GPU DMA transfer (CUDA IPC)
  - Both protocols needed, PCIe limitations serialize DMA across I/O hubs

Integrated Support for User-Defined Datatypes



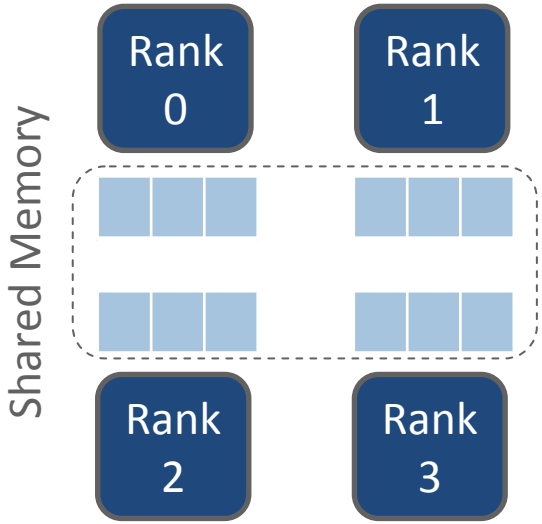


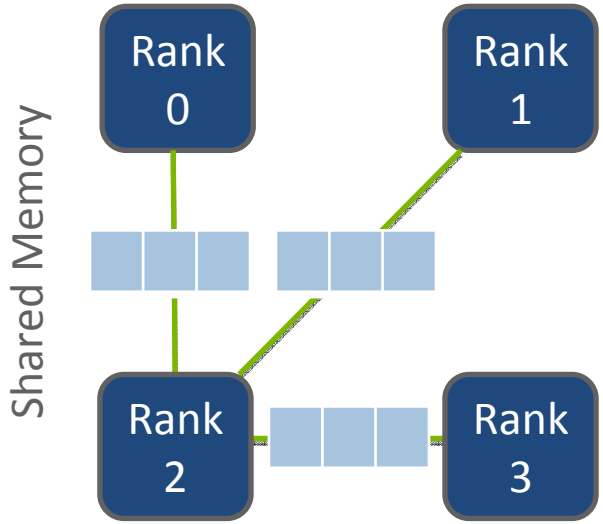
MPI_Send(buffer, ***datatype***, count, to, …)
MPI_Recv(buffer, ***datatype***, count, from, …)

- What if the datatype is noncontiguous?
- CUDA doesn't support arbitrary noncontiguous transfers
- Pack data on the GPU [Jenkins '12]

– Flatten datatype tree representation
– Packing kernel that can saturate memory bus/banks

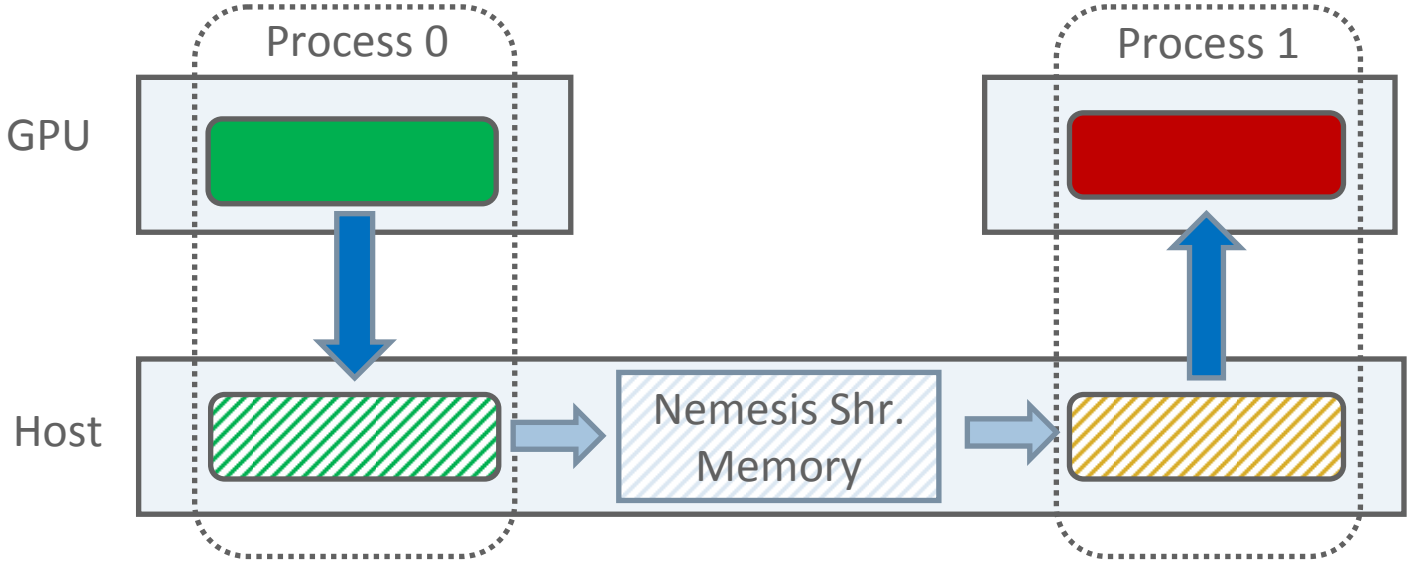
Intranode Communication in MPICH2 (Nemesis)





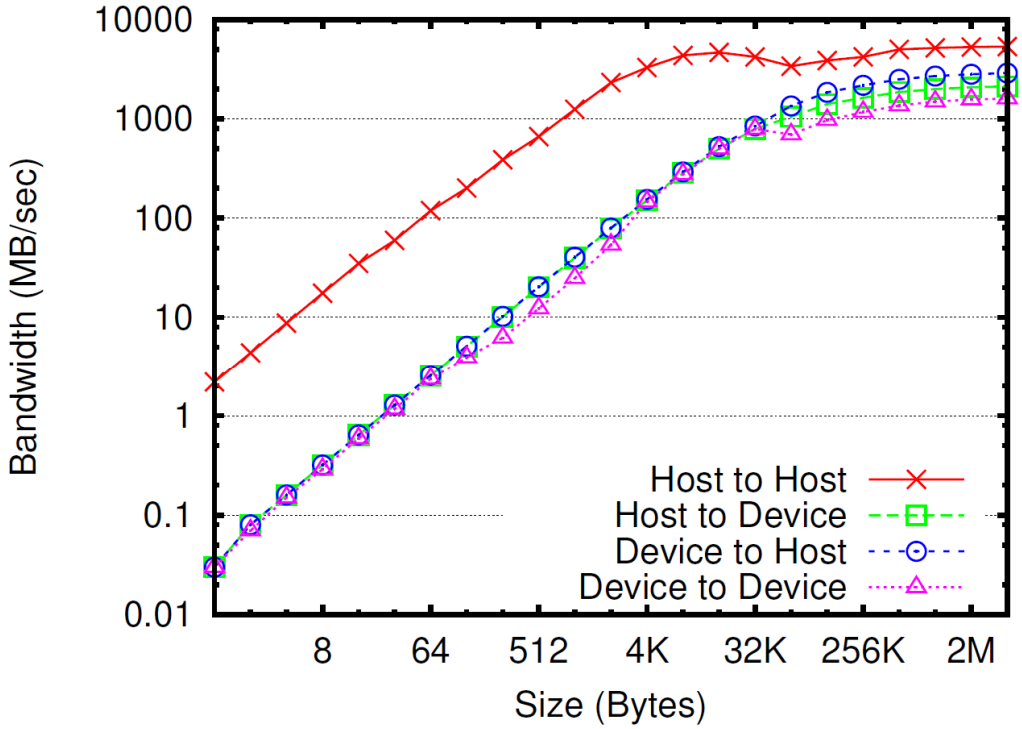
- Short Message Transport
  - Shared message queue
  - Large, persistent queue
  - Single buffer transport
- Large Message Transport
  - Point-to-point shared buffer
  - Ring buffer allocated at first connection
  - Multi-buffered transport

Non-integrated Intranode Communication



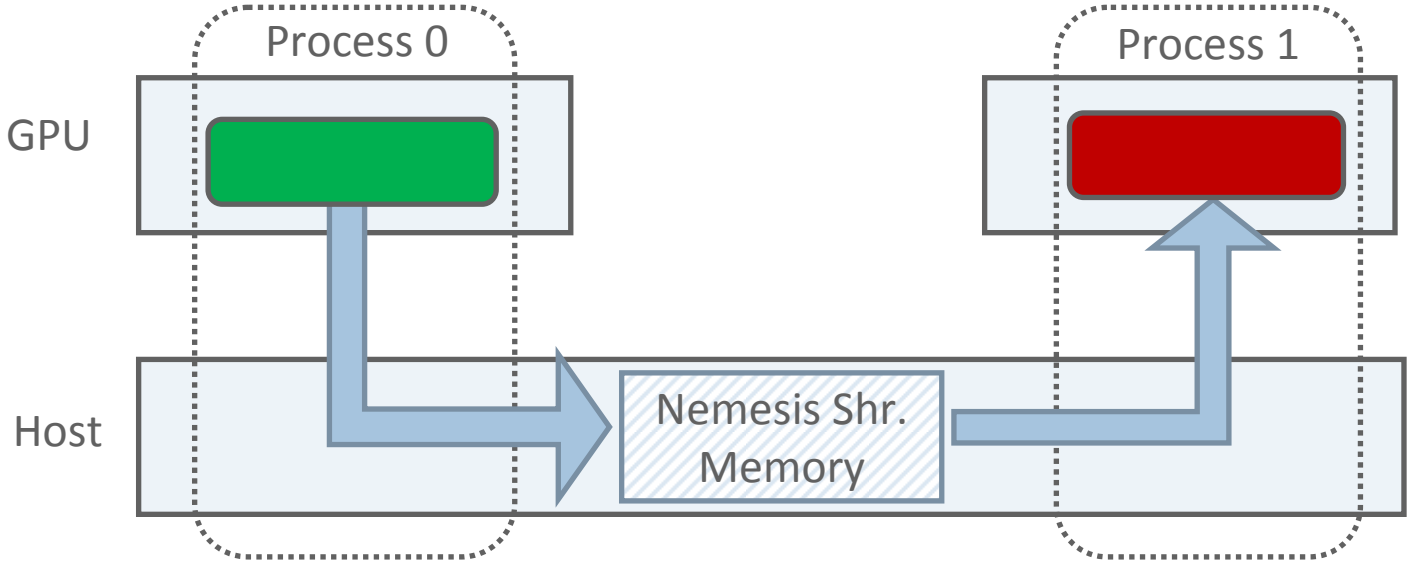
- Communication without accelerator integration
  - 2 PCIe data copies + 2 main memory copies
  - Transfers are serialized

Performance Potential: Intranode Bandwidth



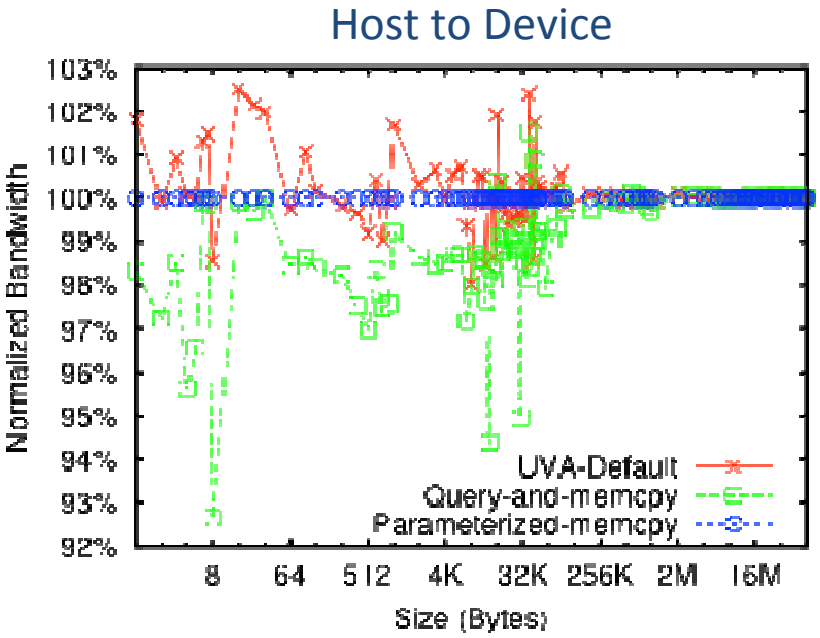
- Bandwidth measurement when using manual data movement
- Theoretical node bandwidth: 6 GB/sec
  - Achieved for host-host transfers
- Observed bandwidth: 1.6 – 2.8 GB/sec
  - With one/two GPU buffers – one/two copies

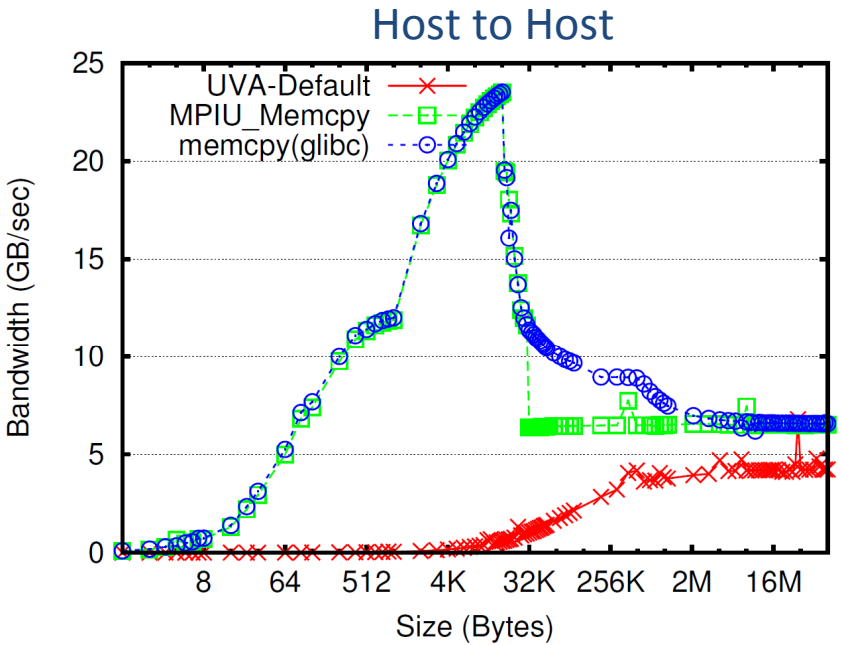
Eliminating Extra Copies



- Integration allows direct transfer into shared memory buffer
- LMT: sender and receiver drive transfer concurrently
  - Pipeline data transfer
  - Full utilization of PCIe links

Copying Data Between Host and Device

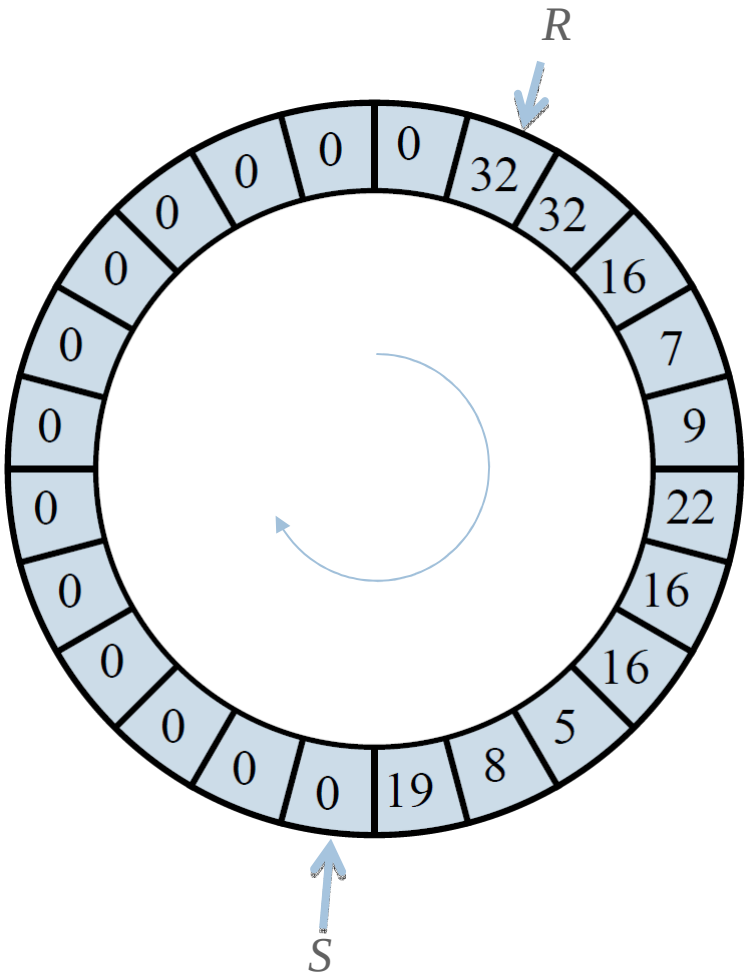




- Three choices for selecting the right copy operation:
  1. *UVA-Default*: Use cudaMemcpy(…, cudaMemcpyDefault)
  2. *Query-and-copy*: UVA query buffer type
     - Dispatch `memcpy` or `cudaMemcpy`
  3. *Parameterized-copy*: Pass parameter for each buffer
     - Dispatch `memcpy` or `cudaMemcpy`

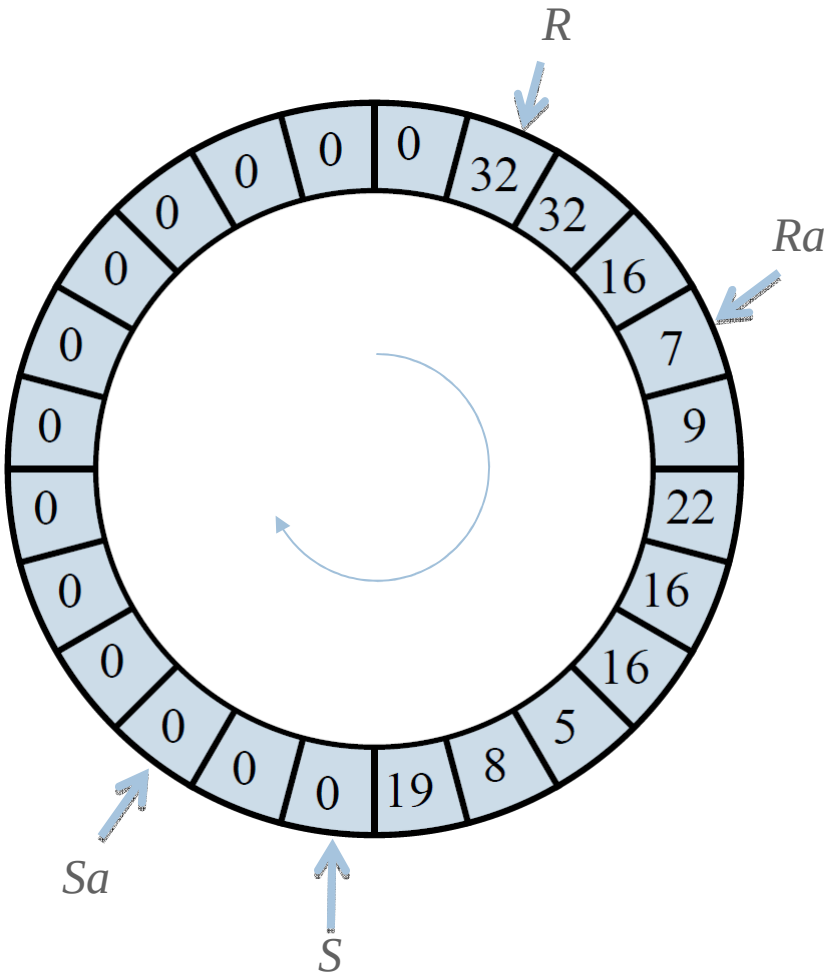
Large Message Transport Protocol

- Shared buffer mapped between pairs of communicating processes
  - Enables pipelined transfer
  - Sender and receiver drive DMA concurrently
- Fixed-size ring Buffer
  - Set of fixed-size partitions
  - *R* - receiver's pointer
  - *S* - sender's pointer
- Partition size
  - Set to data length by Sender
  - Set to zero by Receiver

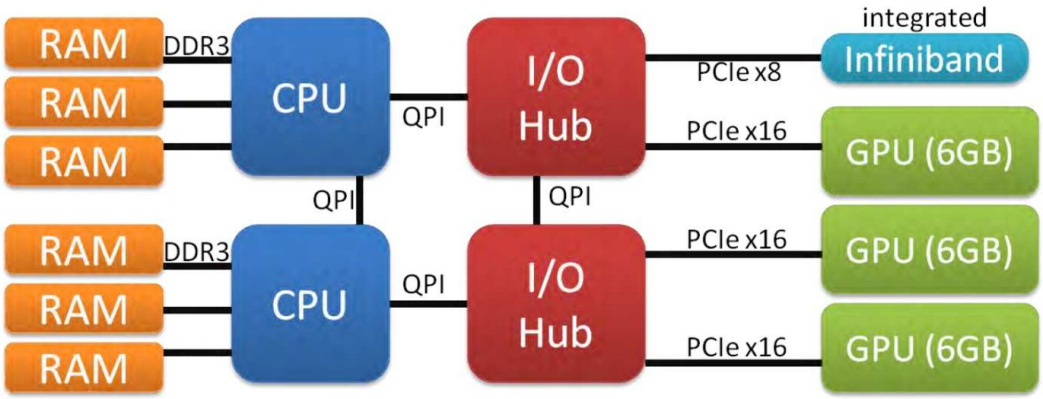


Extending LMT Protocol to Accelerators

- Sender and receiver issue asynchronous PCIe data transfers
  - Add *Ra* and *Sa* pointers
  - Mark section of R/S segment in use by PCIe transfers
- Proactively generate PCIe data transfers
  - Move to the next partition
  - Start new PCIe data copy
  - Repeat until full or RB is empty
  - Update R/S when checking PCIe operations later



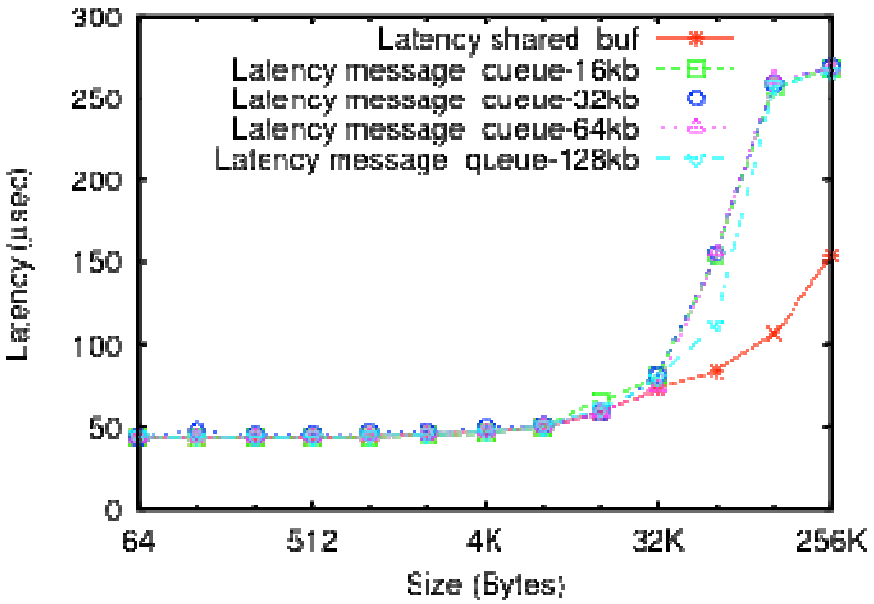
Experimental Evaluation

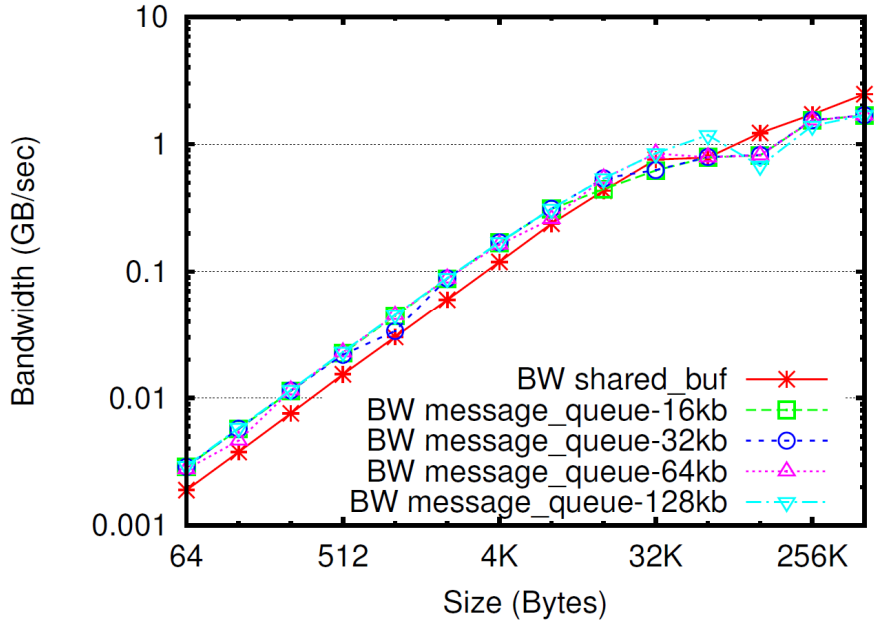


- Keeneland node architecture
  - 2x Intel Xeon X5660 CPUs, 24 GB Memory, 3x Nvidia M2070 GPUs

1. Parameter optimization: shared ring buffer and message queue
2. Communication benchmarking: OSU latency/bw
3. Stencil2D from SHOC benchmark suite

Eager Parameters: Message Queue Element Size and Eager/LMT Threshold



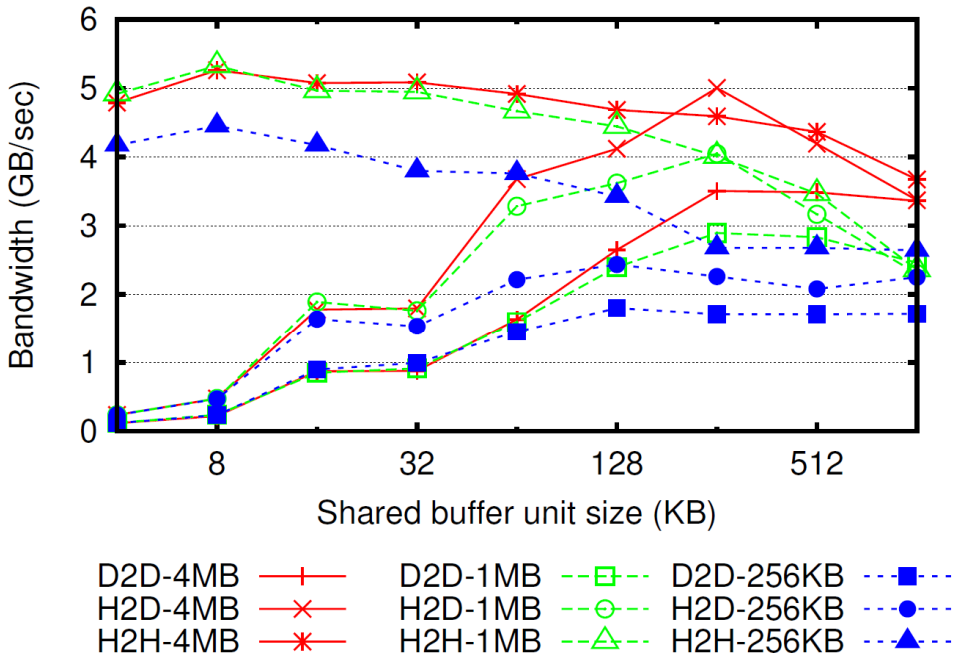


- GPU-GPU transfer of varying size
  - "Shared buf" is manual data transfer
- Large message queue can support more requests
- Eager up to 64 kB, LMT for +64 kB
  - The same value as host-only

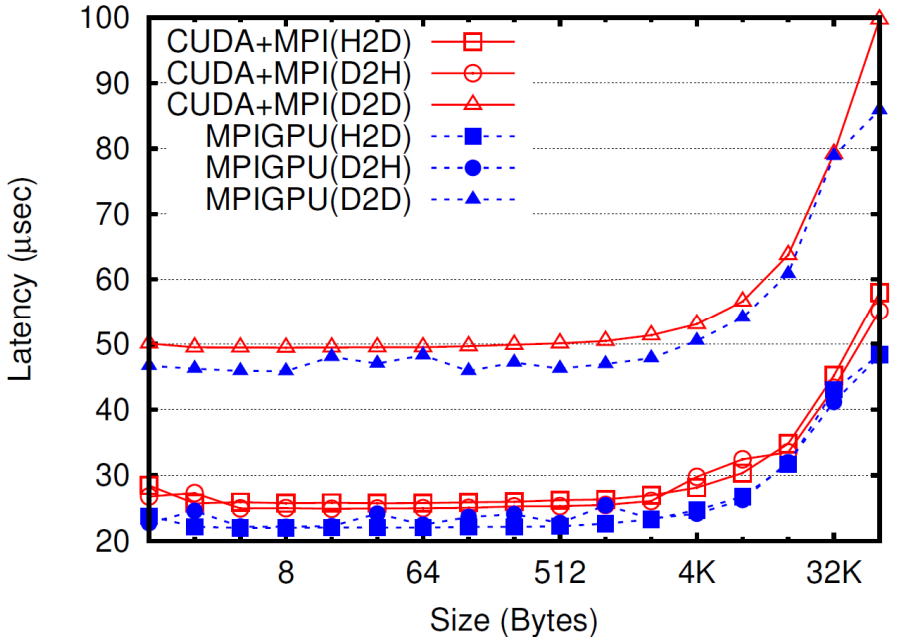
LMT Parameter: Shared Ring Buffer Unit Size

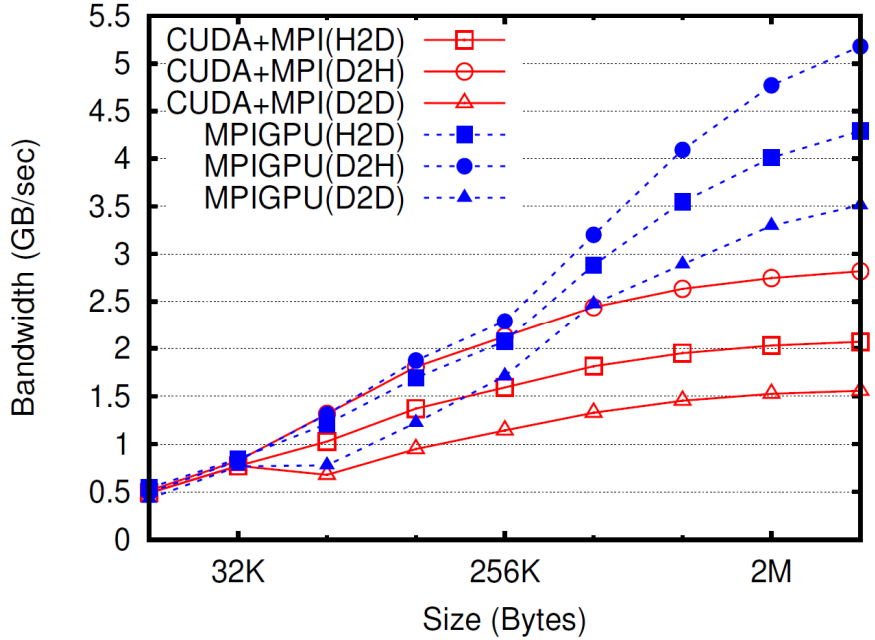
- Host-to-host
  - Default buffer unit size 32 KB
- GPU involved
  - Use 256 KB
  - PCIe bandwidth favors larger messages
- Parameter choice requires knowledge of buffer locations on sender and receiver
  - Exchange information during handshaking phase of LMT protocol

Intranode bandwidth – two processes



Latency & Bandwidth Improvement

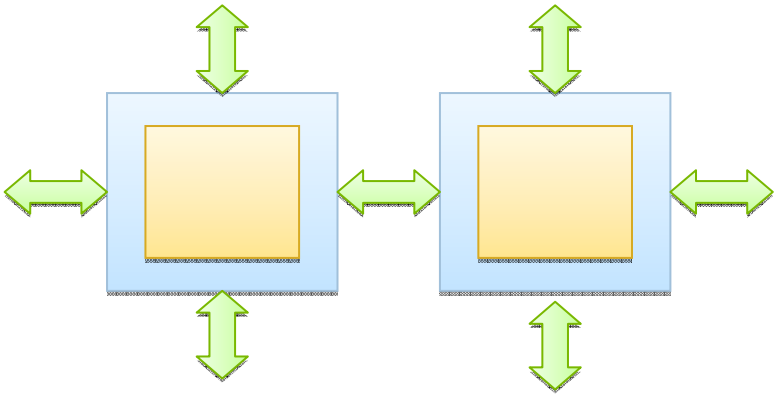


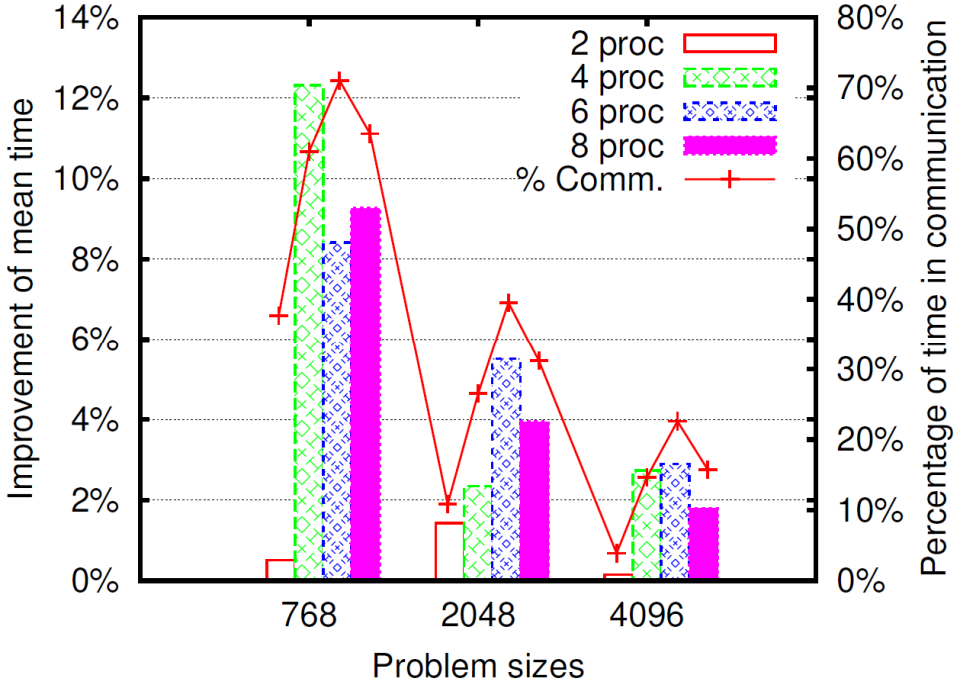


- Less impact on D2D case
  - PCIe latency dominant
- Improvement: 6.7% (D2D), 15.7% (H2D), 10.9% (D2H)

- Bandwidth discrepancy in different PCIe bus directions
- Improvement: 56.5% (D2D), 48.7% (H2D), 27.9% (D2H)
- Nearly saturates peak (6 GB/sec) in D2H case

Application Performance: Stencil2D Kernel



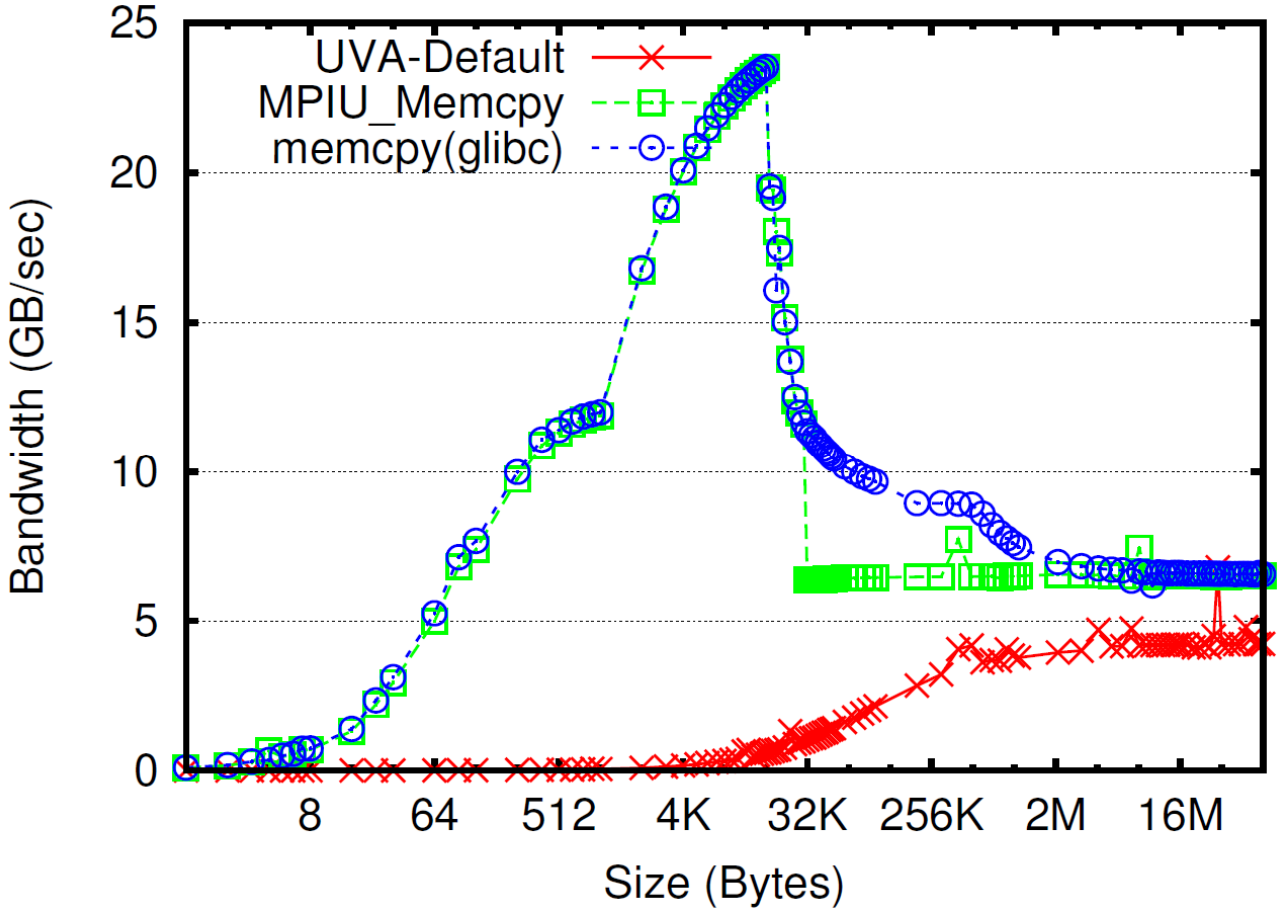


- Nine-point stencil computation, SHOC benchmark suite
  - Halo exchange with neighboring processes
  - Benefit from latency improvement
  - Relative fraction of communication time decreases with problem size
  - Average execution time improvement of 4.3%

Conclusions

- Accelerators are ubiquitous, moving target
  - Exciting new opportunities for systems researchers
  - Requires evolution of HPC software stack
- Integrate accelerator-awareness with MPI
  - Support multiple accelerators and programming models
  - Goals are productivity and performance
- Optimized Intranode communication
  - Eliminate extra main memory copies
  - Pipeline data flow in ring buffer
  - Optimize data copying

Questions?

Backup Slides

Host-side Memcpy Bandwidth