Argonne NATIONAL LABORATORY

MPI-ACC: A Unified Data Movement Infrastructure for Heterogeneous Memory Architectures

Pavan Balaji

Computer Scientist

Argonne National Laboratory

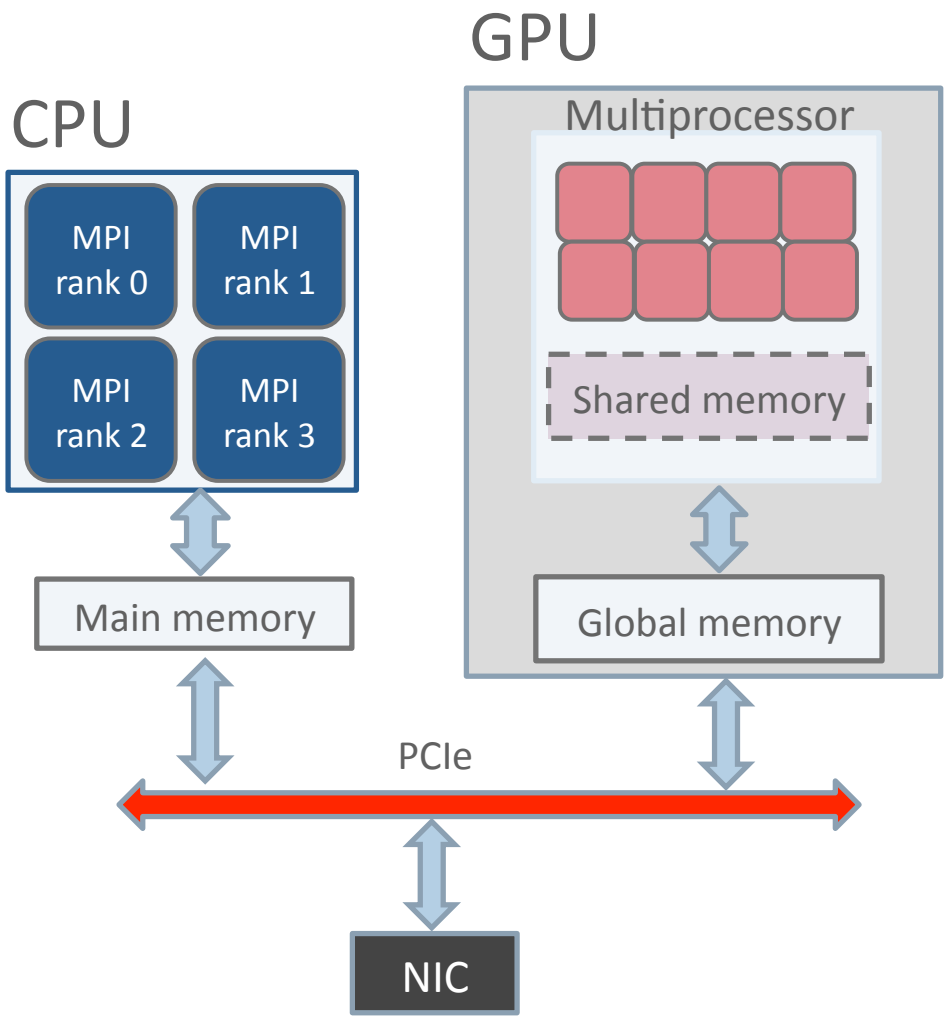
U.S. DEPARTMENT OF ENERGY

Heterogeneous Memory Systems

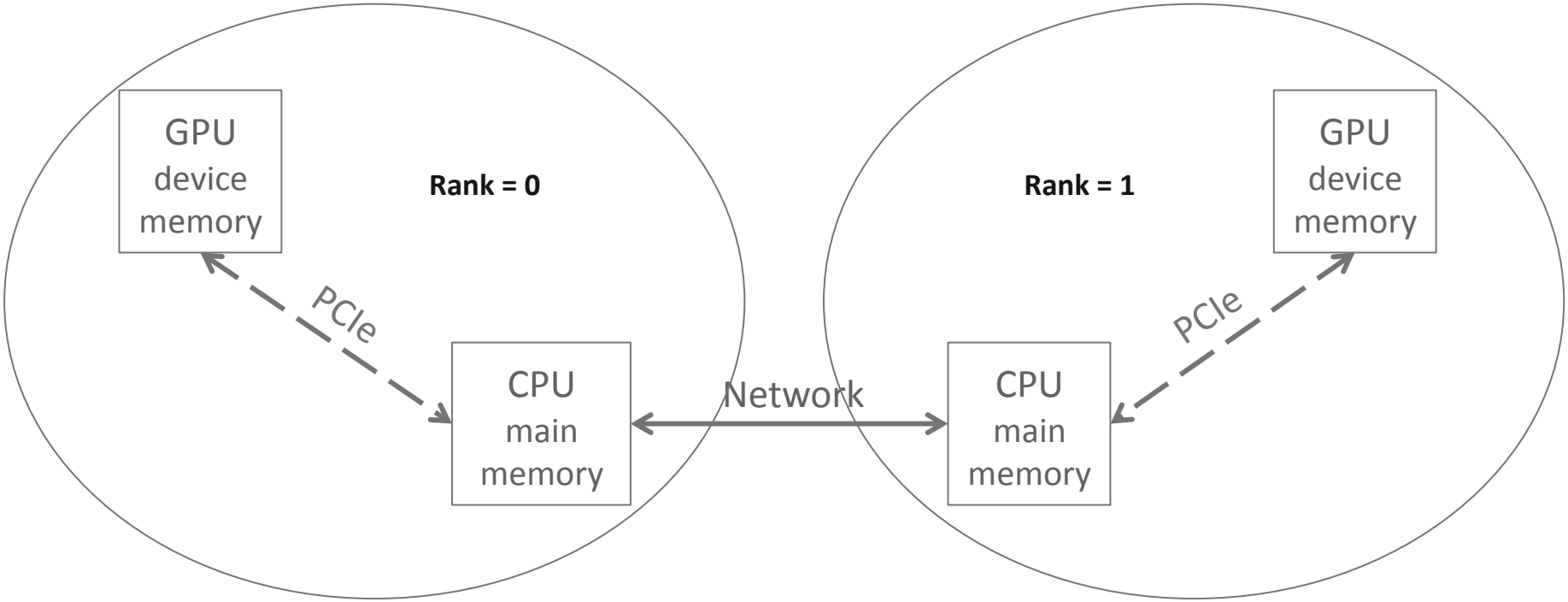
- Increasing focus on heterogeneous memory systems
- Accelerators
  - E.g., Main CPU memory and discrete GPU memory
  - Even with integrated accelerators, memory might still be decoupled
- NVRAM
  - Currently used as a “fast disk” rather than as “bigger memory”
- Scratchpad memory
  - Explicitly managed caches

Example Heterogeneous Architecture: Accelerator Clusters

- Graphics Processing Units (GPUs)
  - Many-core architecture for high performance and efficiency (FLOPs, FLOPs/Watt, FLOPs/$)
  - Prog. Models: CUDA, OpenCL, OpenACC
  - Explicitly managed global memory and separate address spaces
- CPU clusters
  - Most popular parallel prog. model: Message Passing Interface (MPI)
  - Host memory only
- Disjoint Memory Spaces!



Programming Heterogeneous Memory Systems (e.g: MPI+CUDA)




```
if(rank == 0)
{
  cudaMemcpy(host_buf, dev_buf, D2H)
  MPI_Send(host_buf, .. ..)
}
```

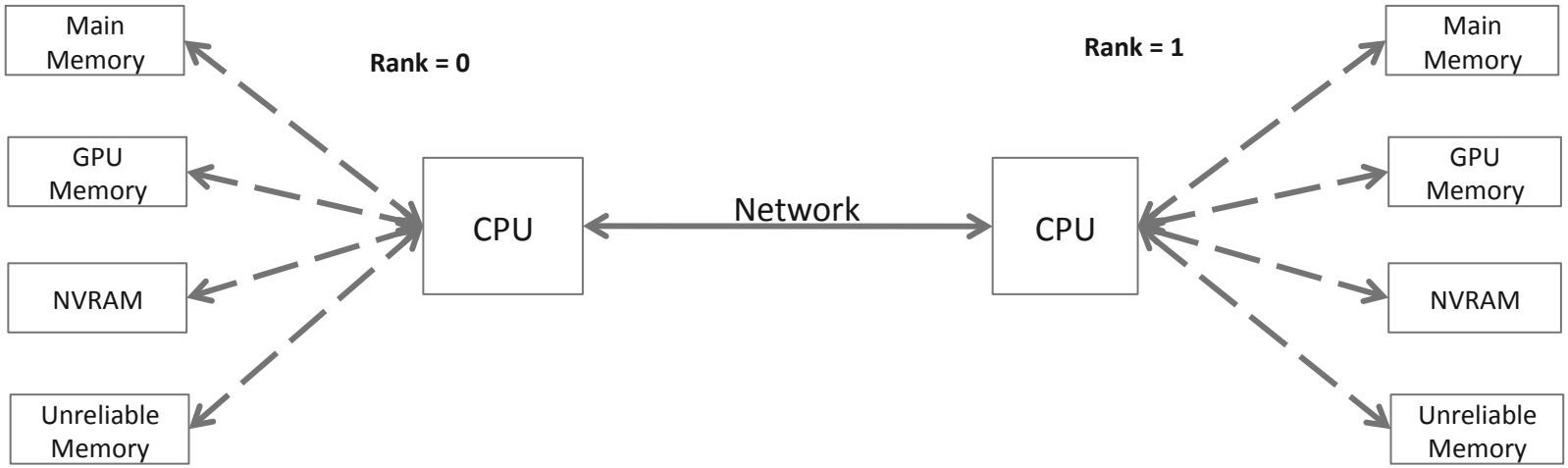
```
if(rank == 1)
{
  MPI_Recv(host_buf, .. ..)
  cudaMemcpy(dev_buf, host_buf, H2D)
}
```

Current Limitations of Programming Heterogeneous Memory Systems (e.g: MPI+CUDA)

```
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);
}
```

- **Programmability/Productivity**: Manual data movement leading to complex code; Non-portable codes
- **Performance**: Inefficient and non-portable performance optimizations
  - Manual copy between host and GPU memory serializes PCIe, Interconnect
    - Difficult for user to do optimal pipelining or utilize DMA engine efficiently
  - Incur protocol overheads multiple times
  - Architecture-specific optimizations

MPI-ACC: A Model for Unified Data Movement



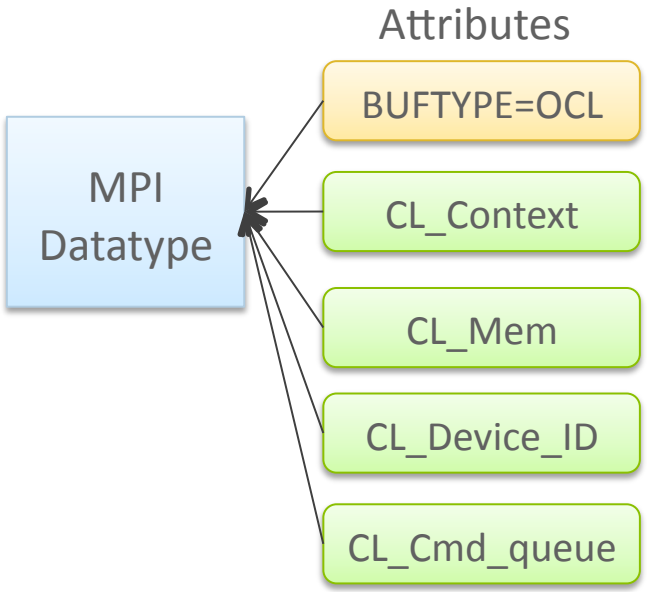

```
if(rank == 0)
{
  MPI_Send(any_buf, .. ..);
}
```

```
if(rank == 1)
{
  MPI_Recv(any_buf, .. ..);
}
```

"MPI-ACC: An Integrated and Extensible Approach to Data Movement in Accelerator-Based Systems", Ashwin Aji, James S. Dinan, Darius T. Buntinas, Pavan Balaji, Wu-chun Feng, Keith R. Bisset and Rajeev S. Thakur. IEEE International Conference on High Performance Computing and Communications (HPCC), 2012

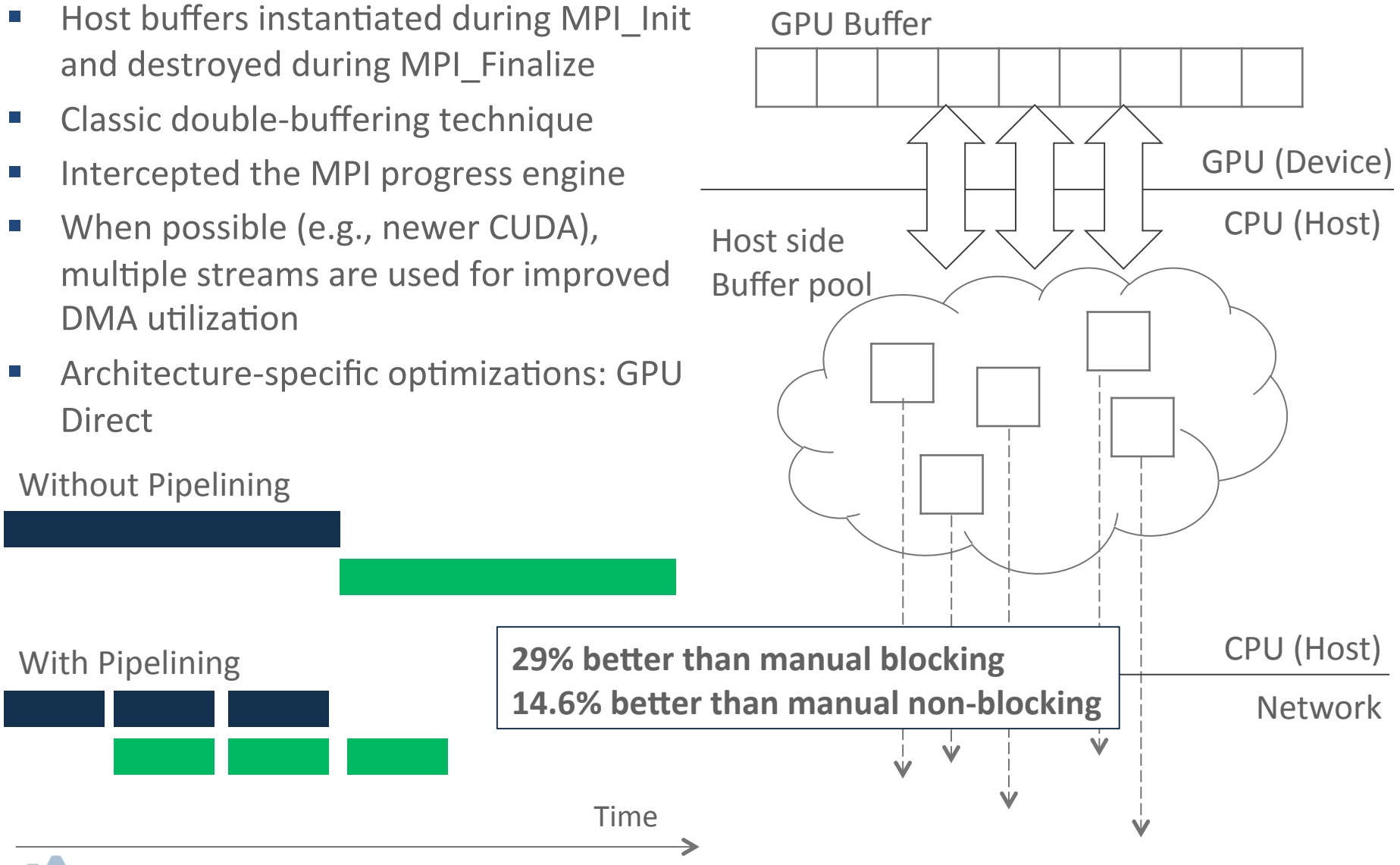
MPI-ACC: Programming Interface

- Different programming models possible:
  - Unified Virtual Address
    - Fully transparent
    - Only possible for some memory architectures
  - Extended function name space
    - MPI_SEND_ACC(…, MPI_ACC_CUDA_MEMORY)
    - Might be slightly better for C++ with function overloading
  - Derived datatype attributes
    - Attached attributes to specify where the data resides
  - Special communicator attributes
    - Collectives that use both host memory and other memory regions not possible
  - Special tag space
    - Not useful for functions that do not use tags

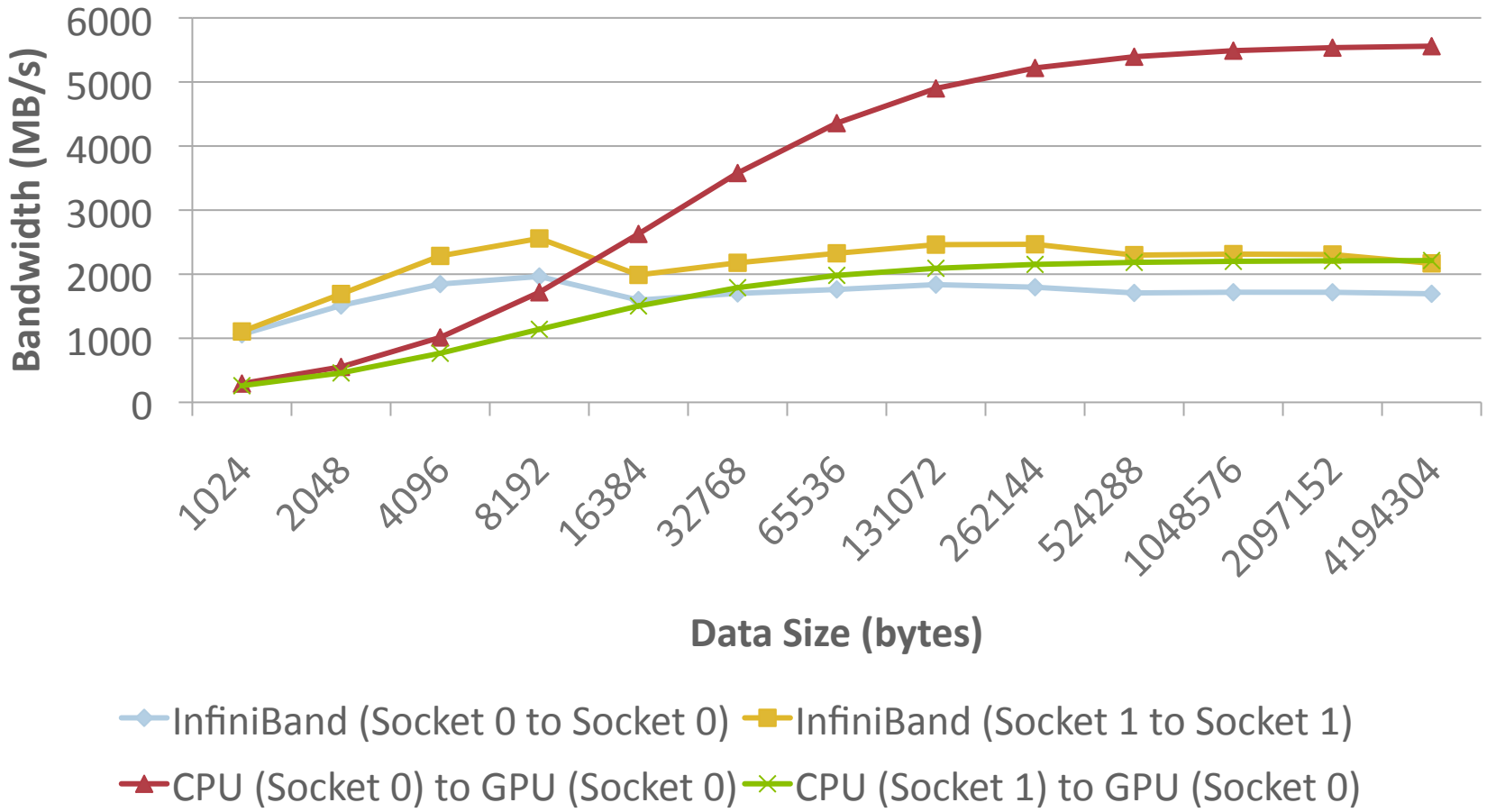


MPI-ACC Optimizations: Pipelined Data Movement

- Host buffers instantiated during MPI_Init and destroyed during MPI_Finalize
- Classic double-buffering technique
- Intercepted the MPI progress engine
- When possible (e.g., newer CUDA), multiple streams are used for improved DMA utilization
- Architecture-specific optimizations: GPU Direct



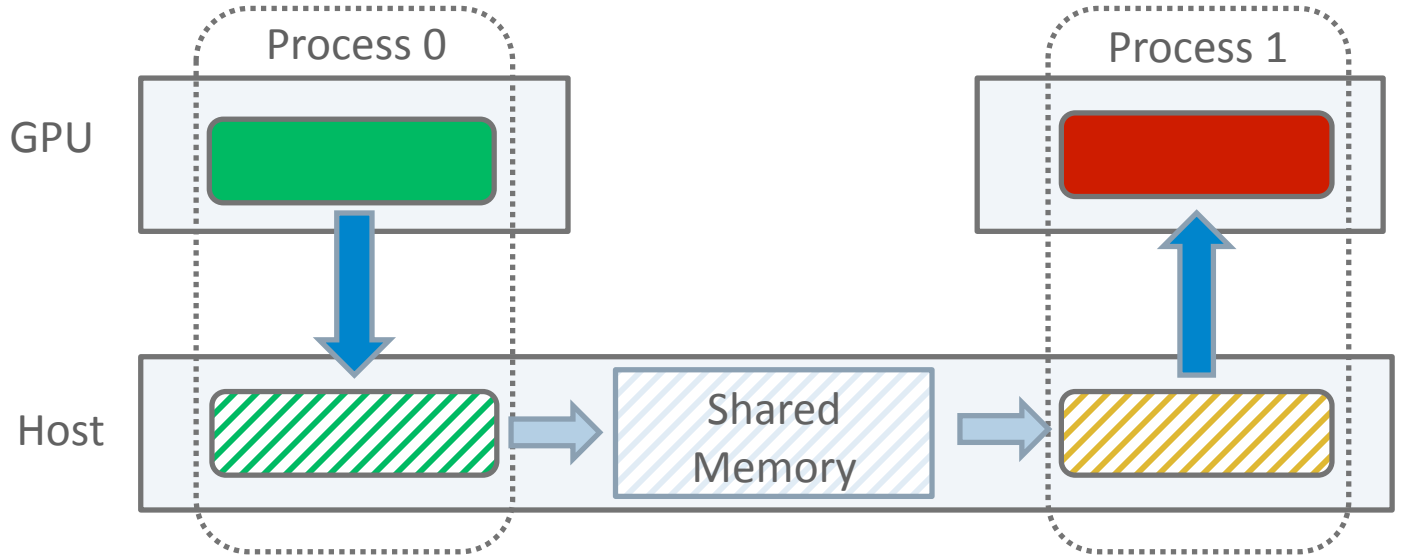
Topology Aware Optimizations



Handle Caching

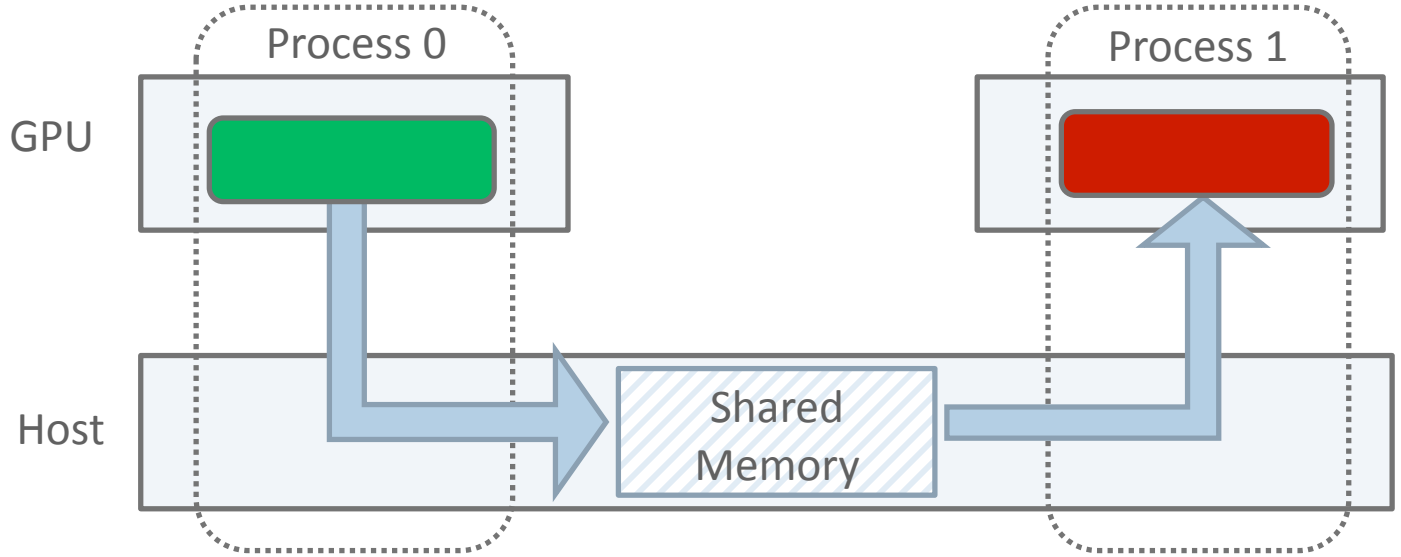
- Applications have a handle to move computation or data to accelerators (stream in CUDA; command queue in OpenCL)
- On some systems, aggregate performance using multiple queues is better than performance using a single queue
- Ideally we would like to use 2-4 queues for large data transfers, instead of a single queue
  - Problem: creating extra queues at data transfer time is expensive
- We internally use multiple queues that are created at the first data movement time and cached
  - Improves performance by a few percent
  - NVIDIA engineers promised that they'll fix this in hardware
  - AMD engineers promised that they'll not

Traditional Intranode Communication



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

Eliminating Extra Copies

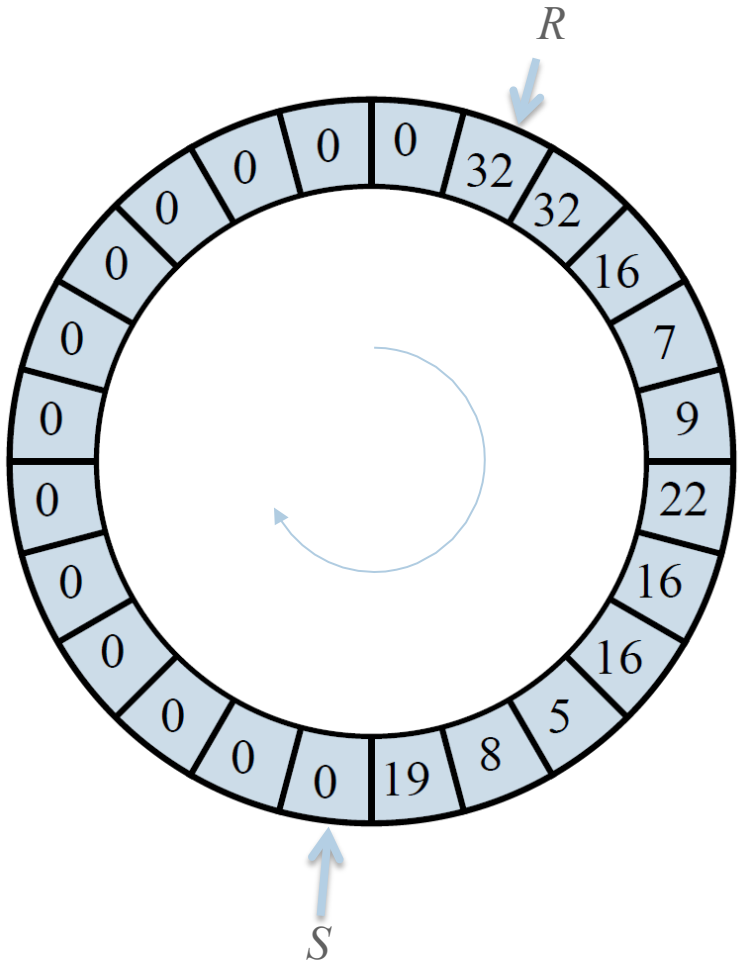


- Integration allows direct transfer into shared memory buffer
- LMT: sender and receiver drive transfer concurrently
  - Pipeline data transfer
  - Full utilization of PCIe links

"Optimizing GPU-to-GPU intra-node communication in MPI", Feng Ji, James S. Dinan, Darius T. Buntinas, Pavan Balaji, Xiaosong Ma and Wu-chun Feng. Workshop on Accelerators and Hybrid Exascale Systems (AsHES); in conjunction with the IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2012

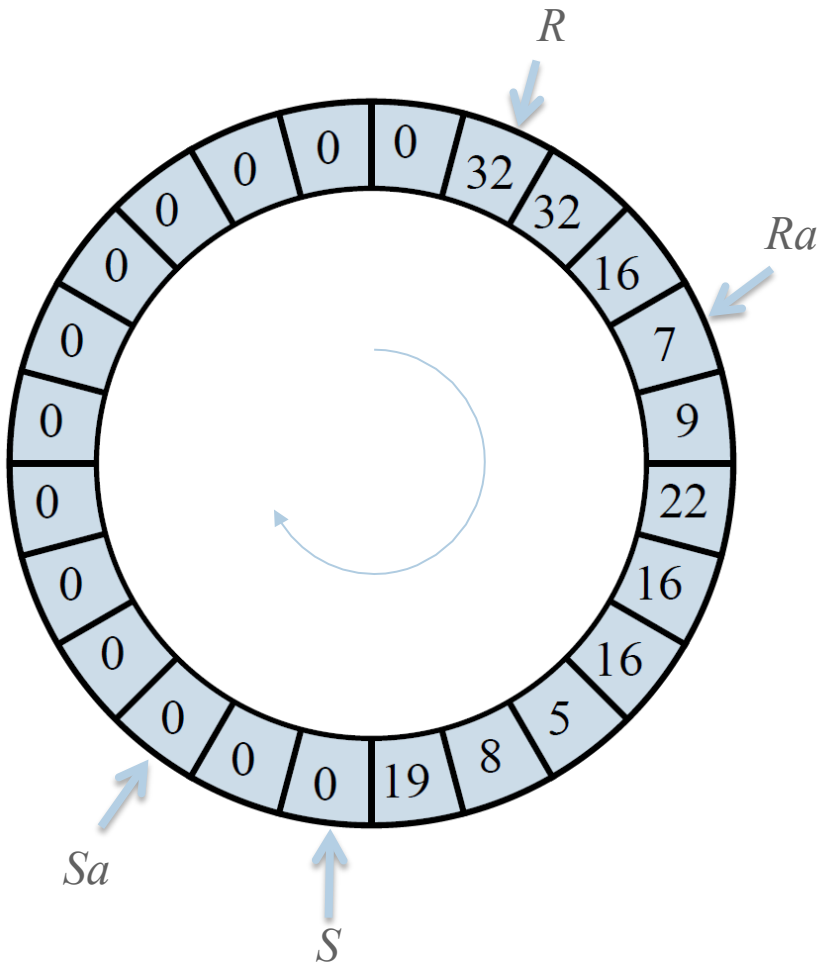
Large Message Transport Protocol

- 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

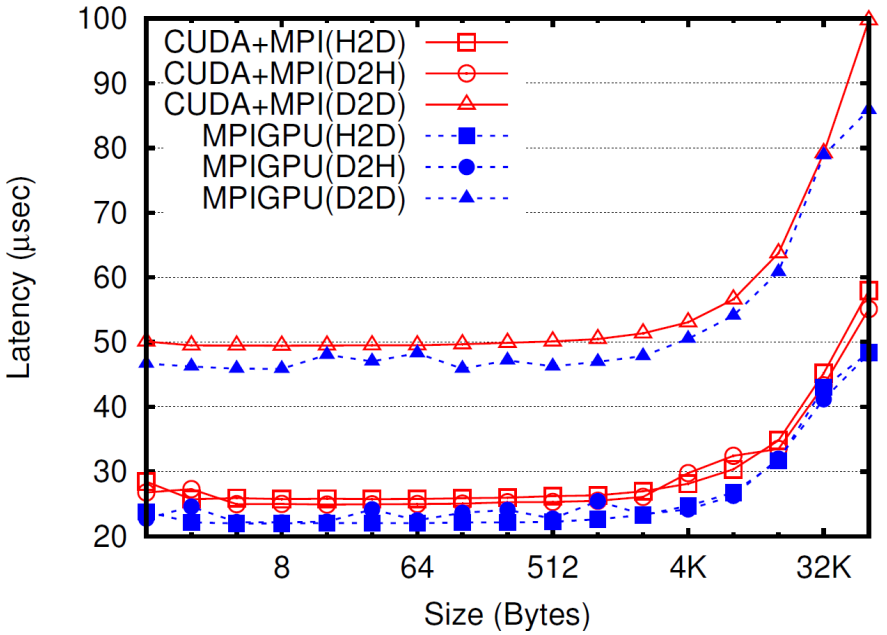


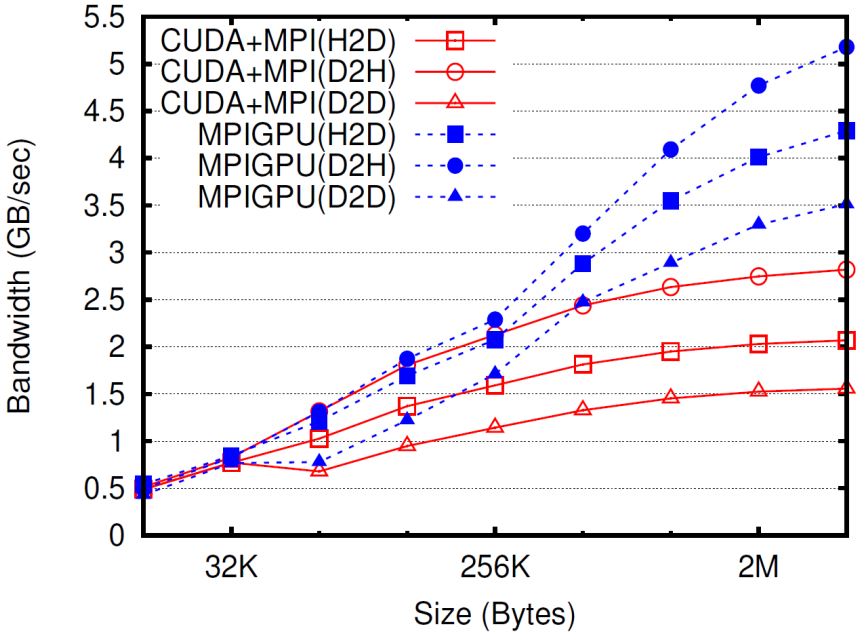
Extending LMT Protocol to Accelerators

- 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



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

GPU Direct and CUDAIPC optimizations

- GPUDirect: DMA-driven peer GPU copy
- CUDAIPC: exporting a GPU buffer to a different process

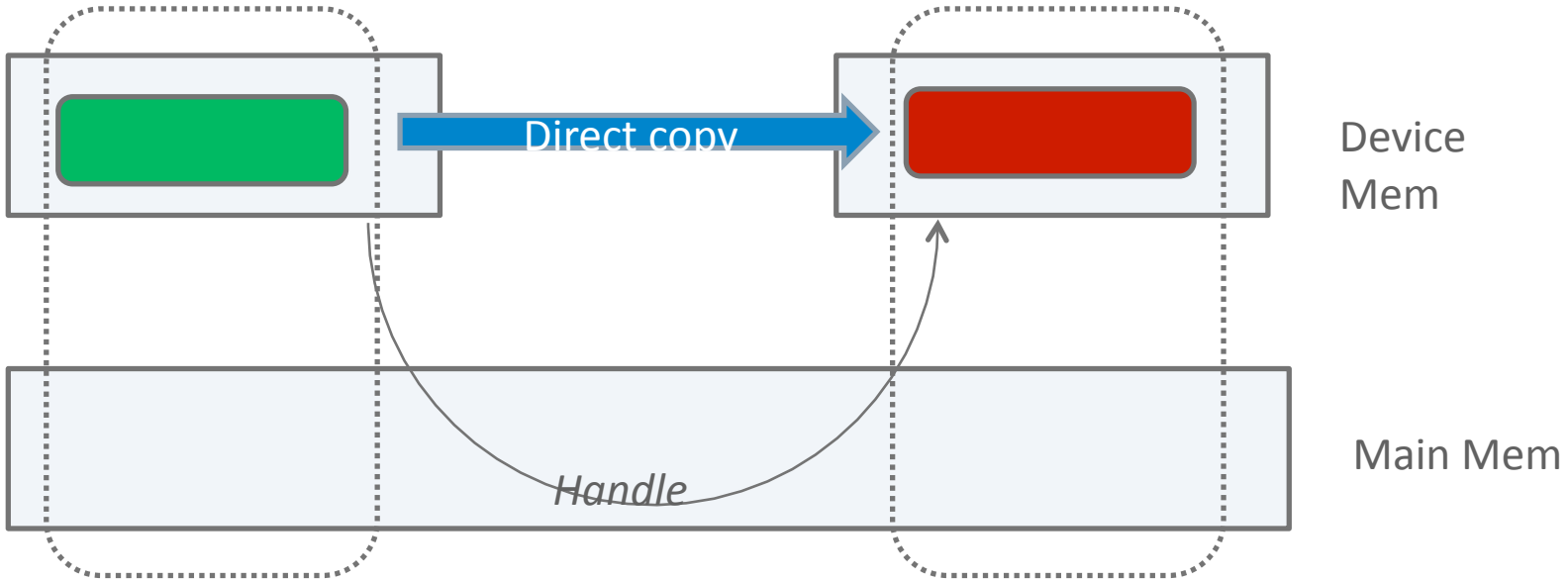
Process 0

cudaIpcGetMemHandle(&*handle*, d_ptr);

Process 1

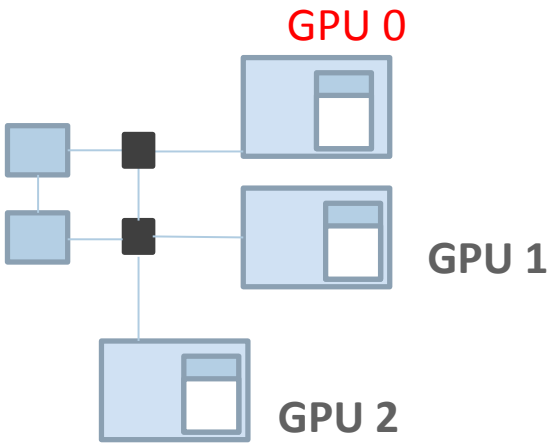
cudaIpcOpenMemHandl(&d_ptr_src, *handle*);

cudaMemcpy(d_ptr, d_ptr_src, …);

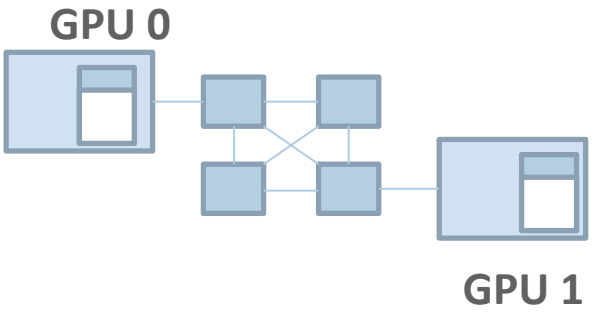


DMA-assisted Intranode GPU data transfer

- Challenges
  - GPUDirect requires GPU *peer accessibility*
    - Same IO/Hub: *Yes*
    - Different IO/Hub: *Yes* for AMD (HT); *No* for Intel (QPI)
  - Overhead of handle open/close



Intel (with QPI)



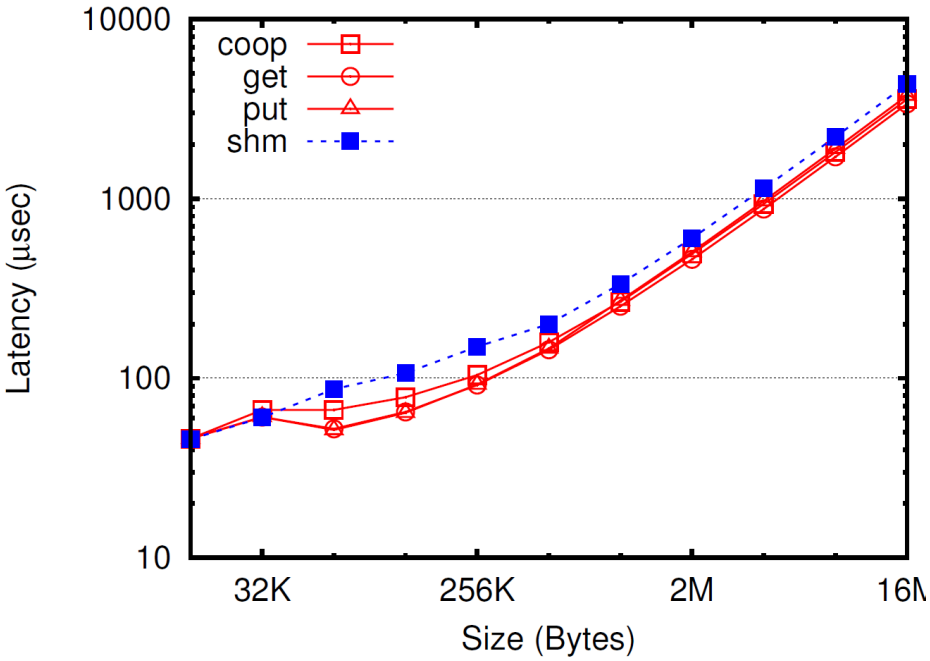
AMD (with HT)

IPC Open/Close overhead

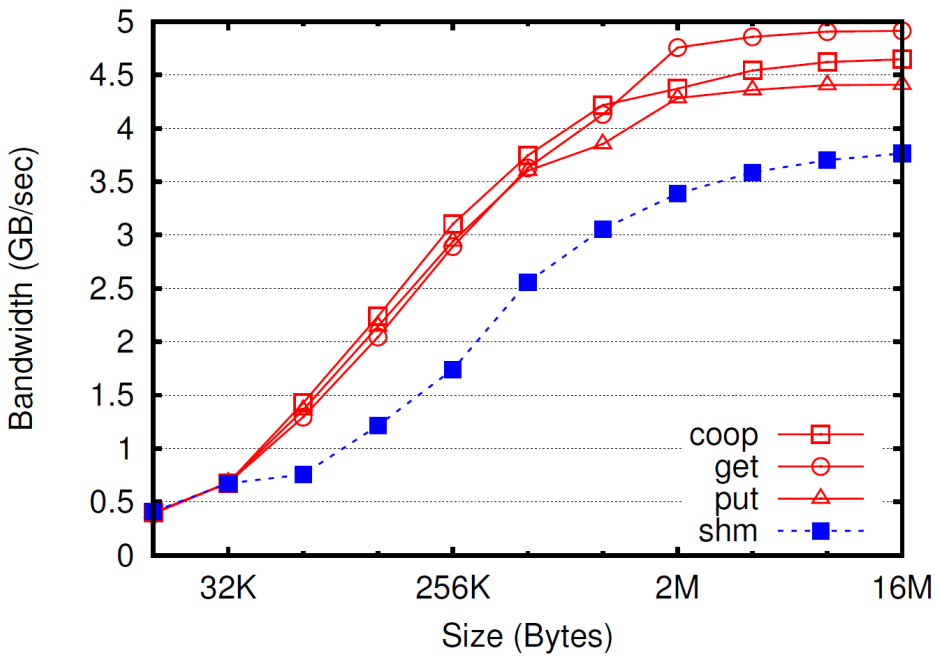
- Getting a handle is light-weight operation
- Open and close operations on the handle are NOT
- Do not close a handle when a pair of MPI_Send/Recv is done
  - Many MPI program reuse buffers (including GPU buffers)
  - Lazy *close* option to avoid overhead
- Cache opened handles & their addresses locally
- Next time: try to find the handle in the local cache
  - If found, no need to reopen it, but use its address

Near case: Keeneland

- Bandwidth nearly reaches the peak bandwidth of the system



(a) Latency versus Message Size

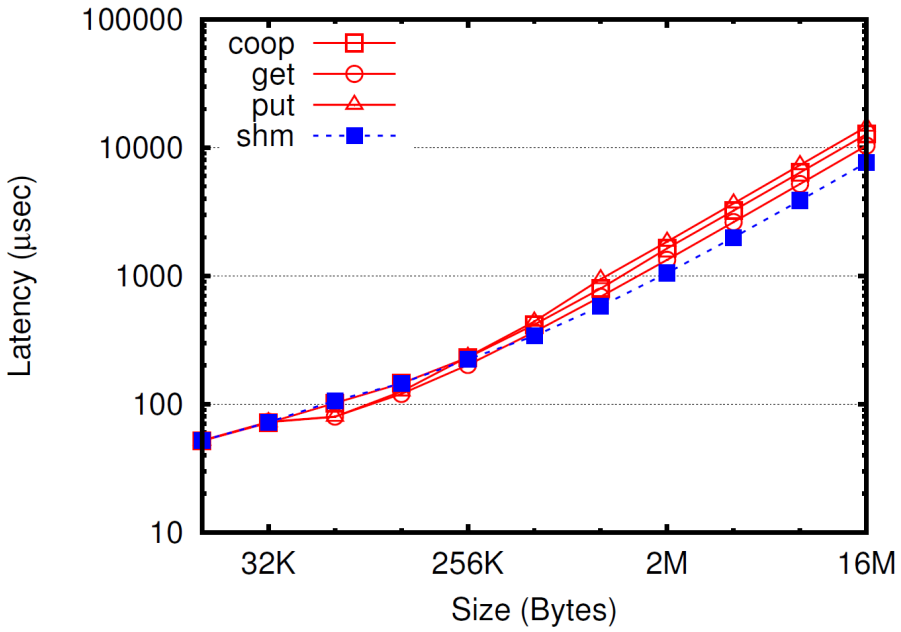


(b) Bandwidth versus Message Size

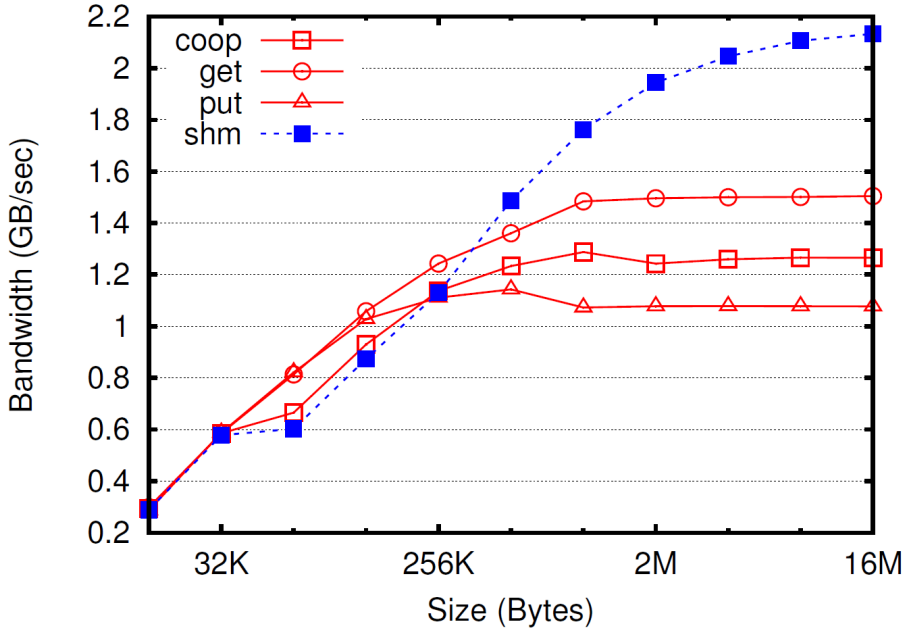
"DMA-Assisted, Intranode Communication in GPU Accelerated Systems", Feng Ji, Ashwin Aji, James S. Dinan, Darius T. Buntinas, Pavan Balaji, Rajeev S. Thakur, Wu-chun Feng and Xiaosong Ma. IEEE International Conference on High Performance Computing and Communications (HPCC), 2012

Far case: Magellan

- Better to adopt shared memory approach

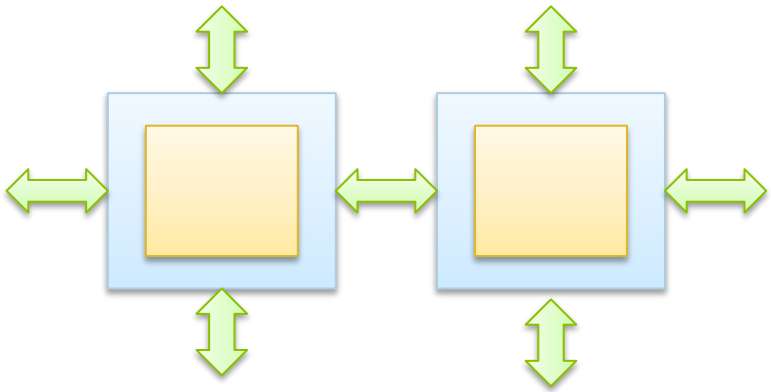


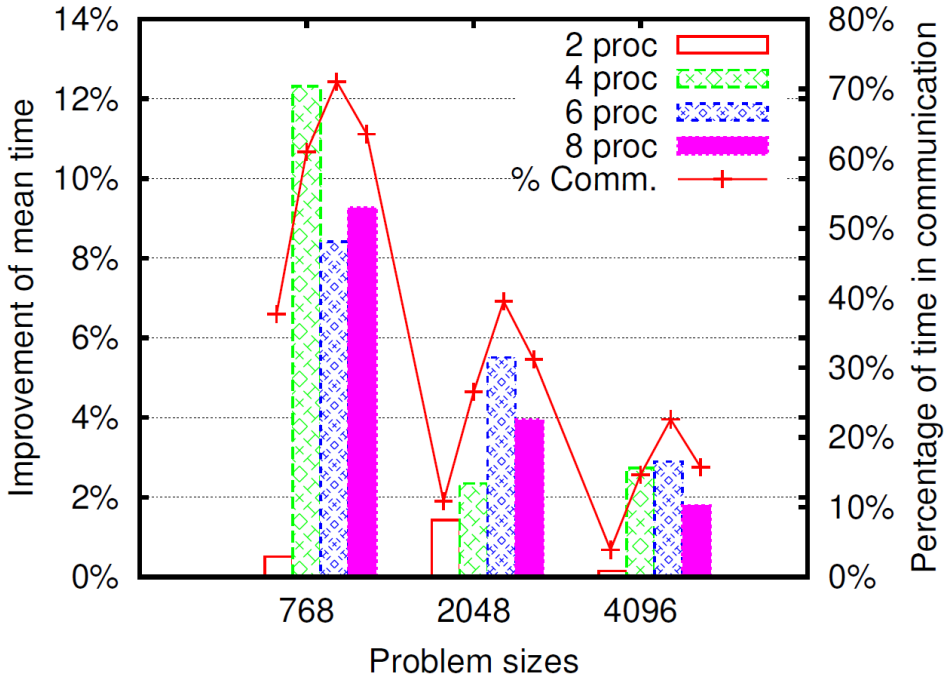
(a) Latency versus Message Size



(b) Bandwidth versus Message Size

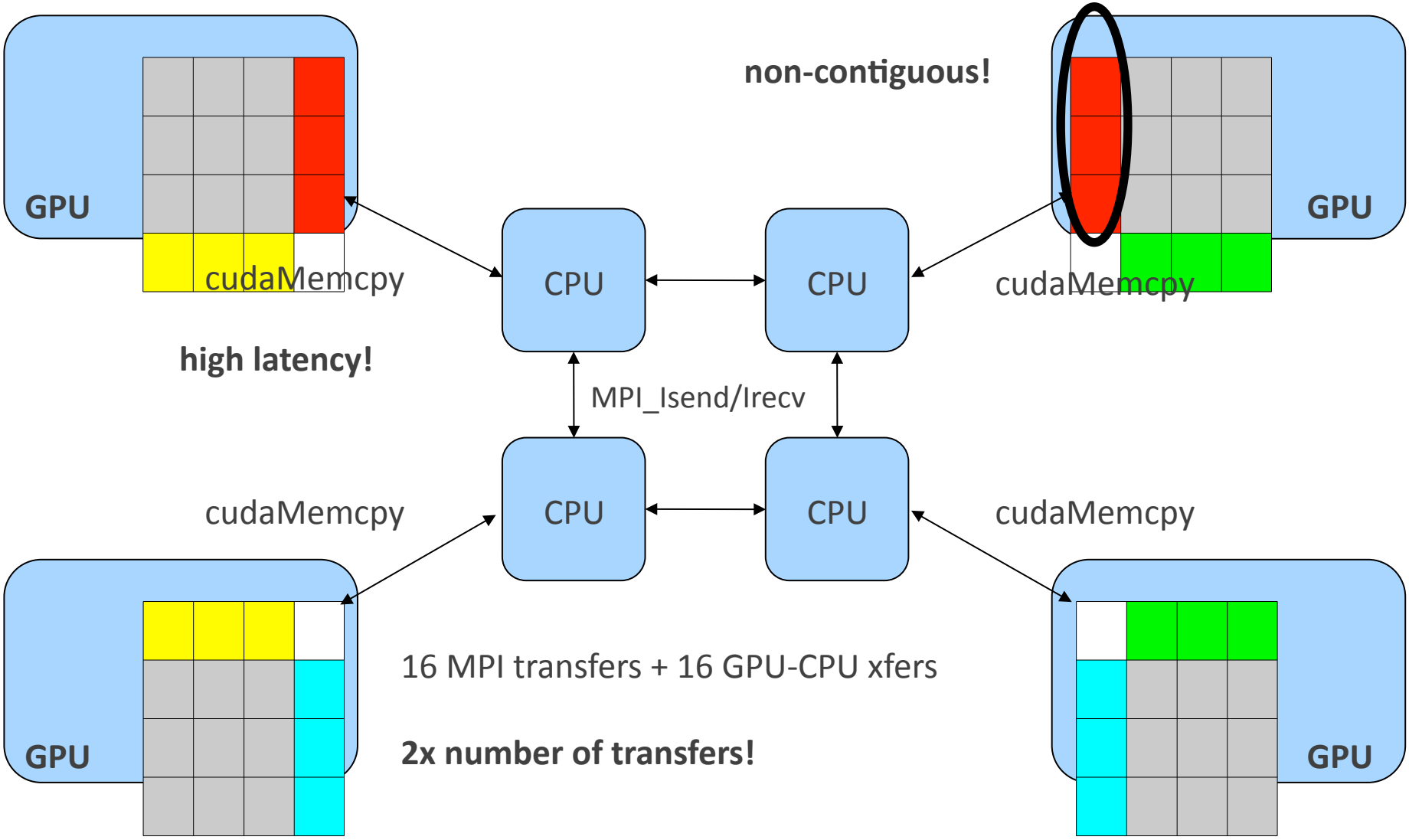
Application Performance: Stencil2D Kernel





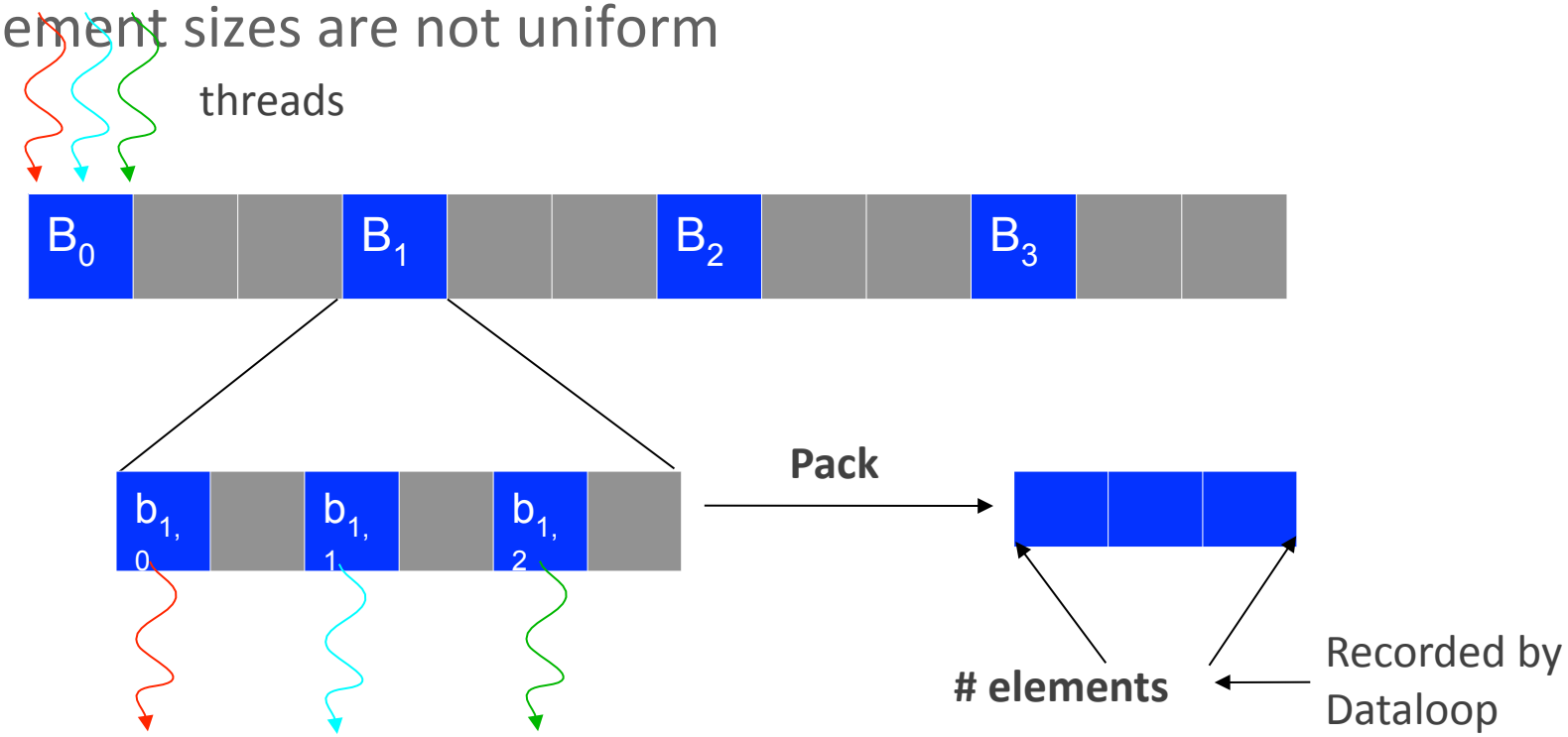
- Nine-point stencil computation, SHOC benchmark suite
  - Halo exchange with neighboring processes
  - Benefit from latency improvement
  - Relative fraction of communication time decreases with problem size
  - Average execution time improvement of 4.3%

MPI + GPU Example – Stencil Computation



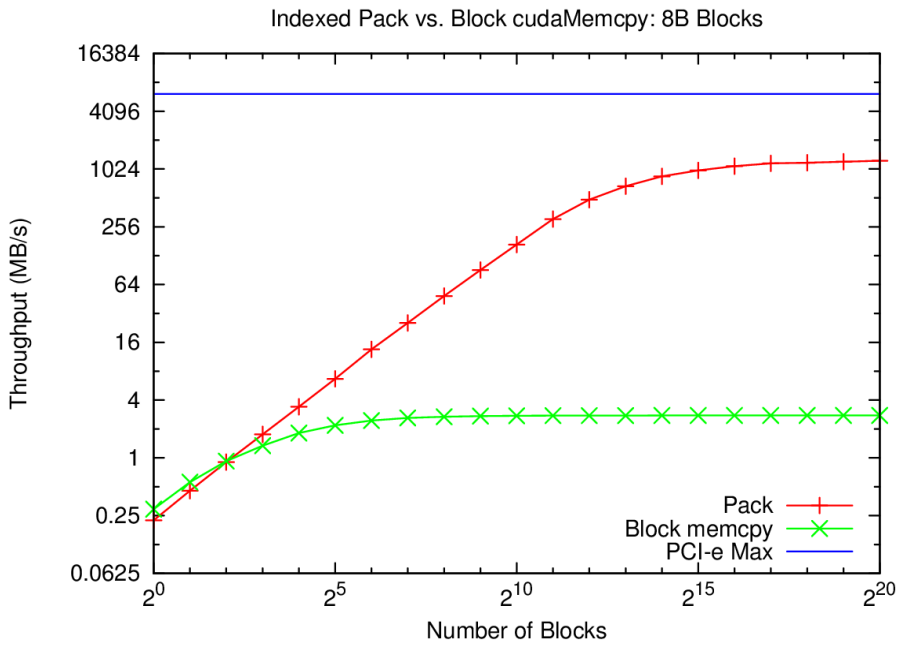
GPU optimizations for Data Packing

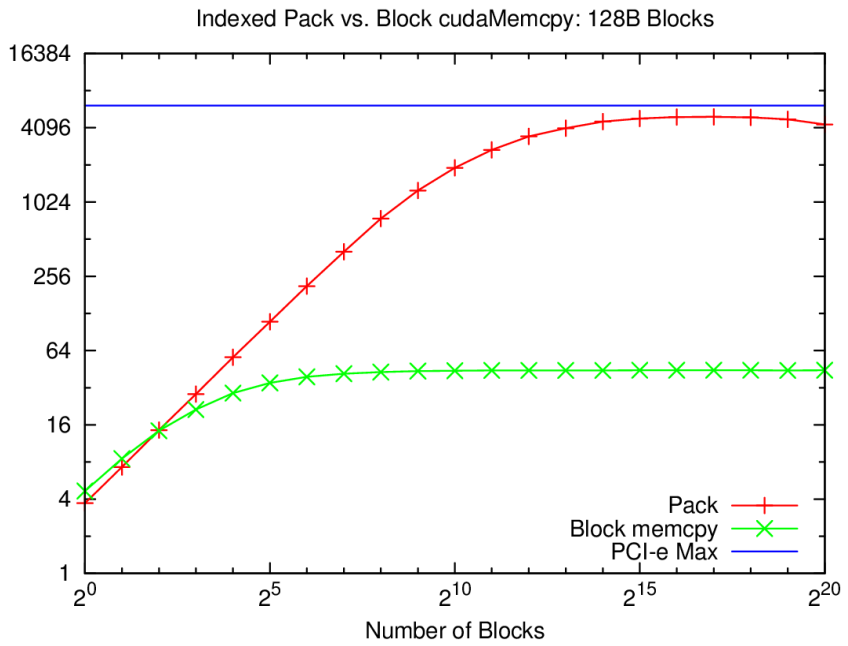
- Element-wise traversal by different threads
- Embarrassingly parallel problem, except for structs, where element sizes are not uniform

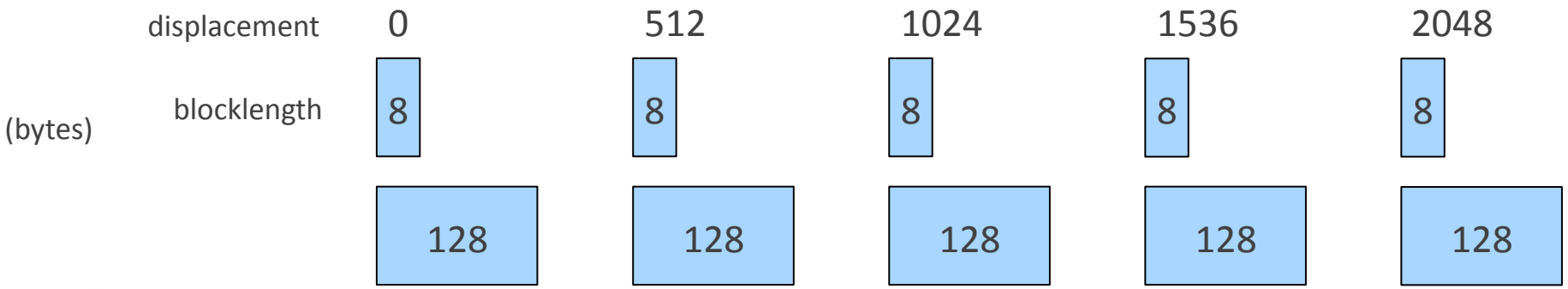


traverse by **element #**, read/write using **extent/size**

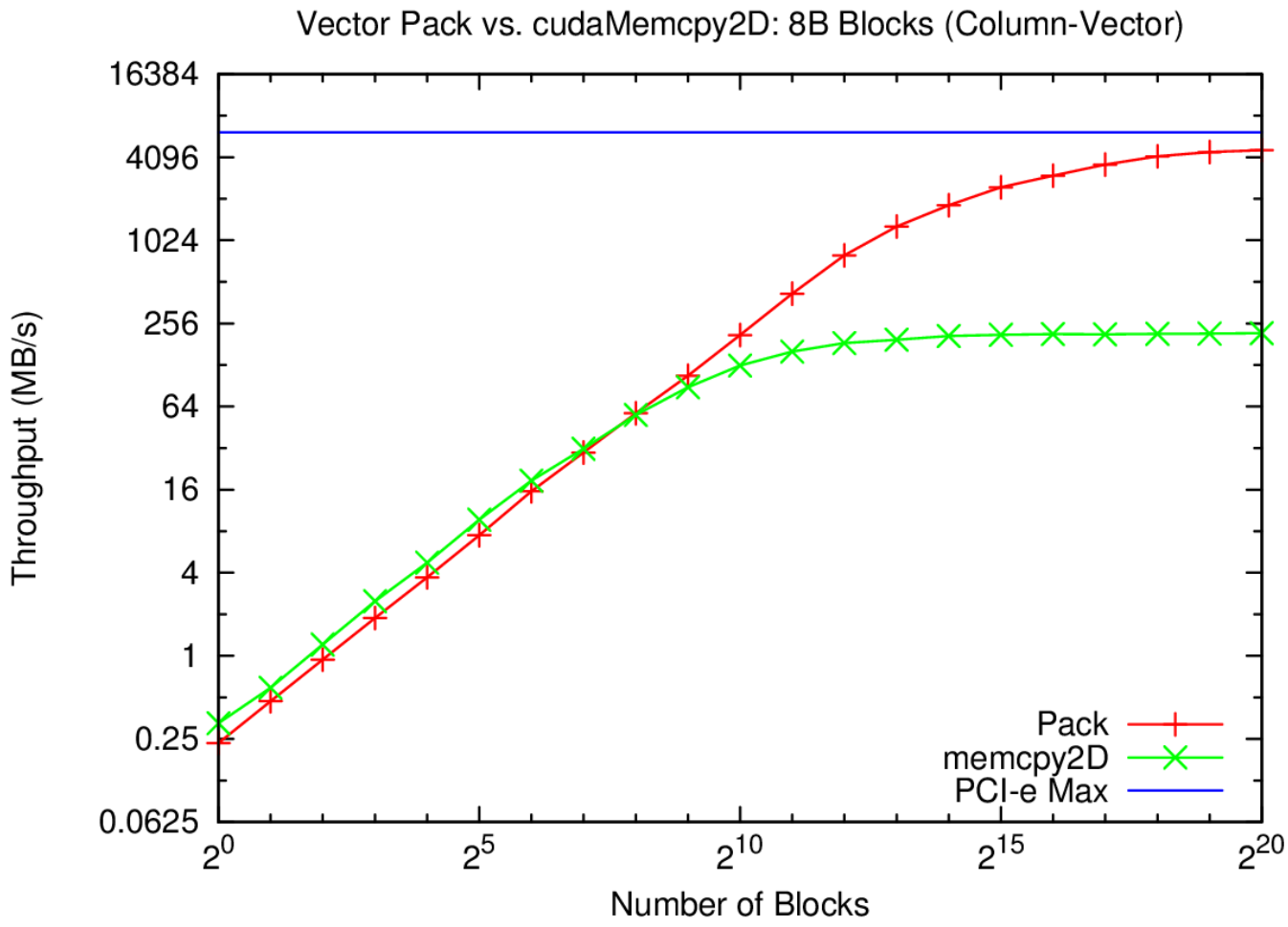
Packing Throughput (Indexed)





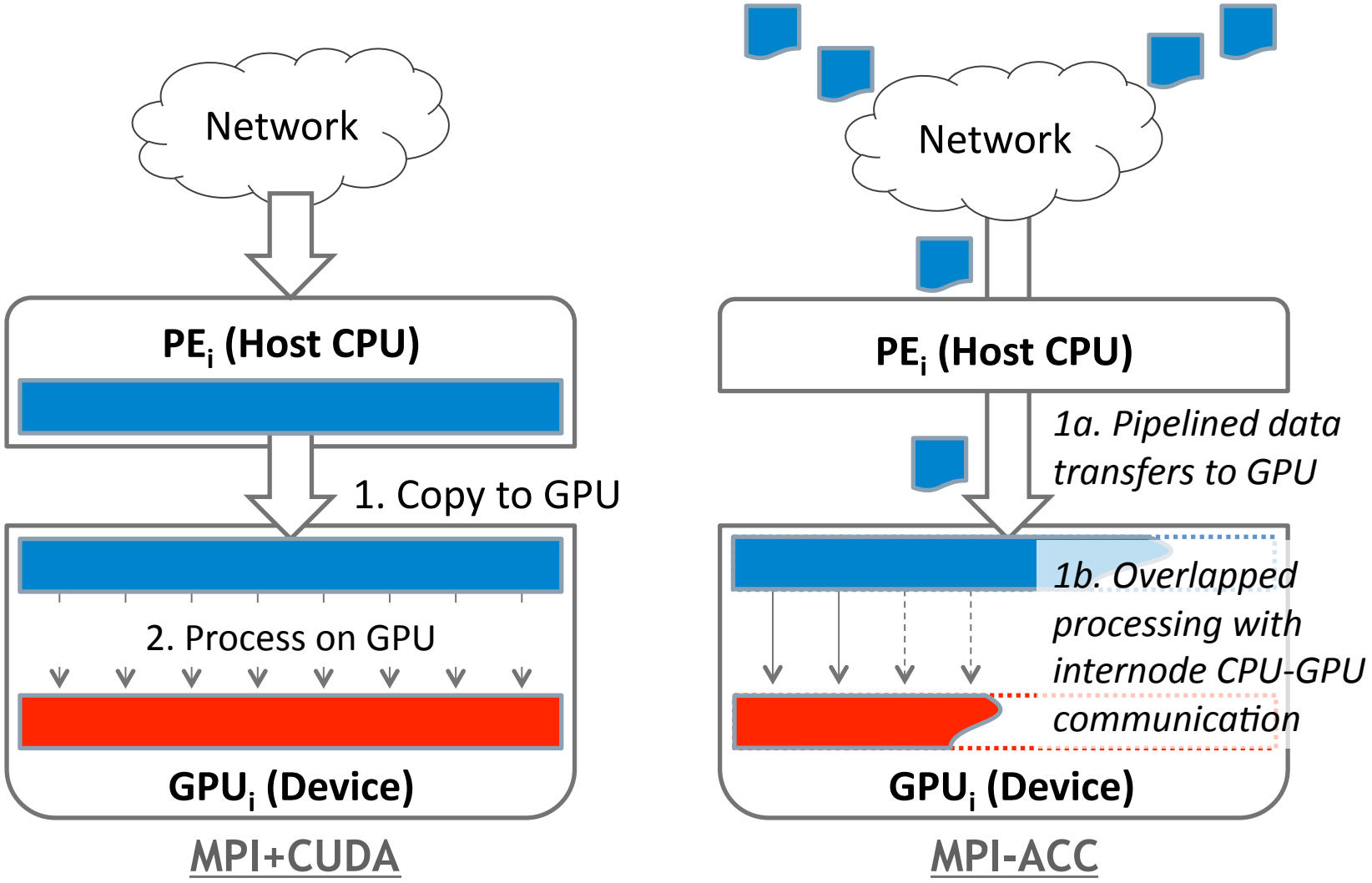


Packing Throughput (Column-Vector)

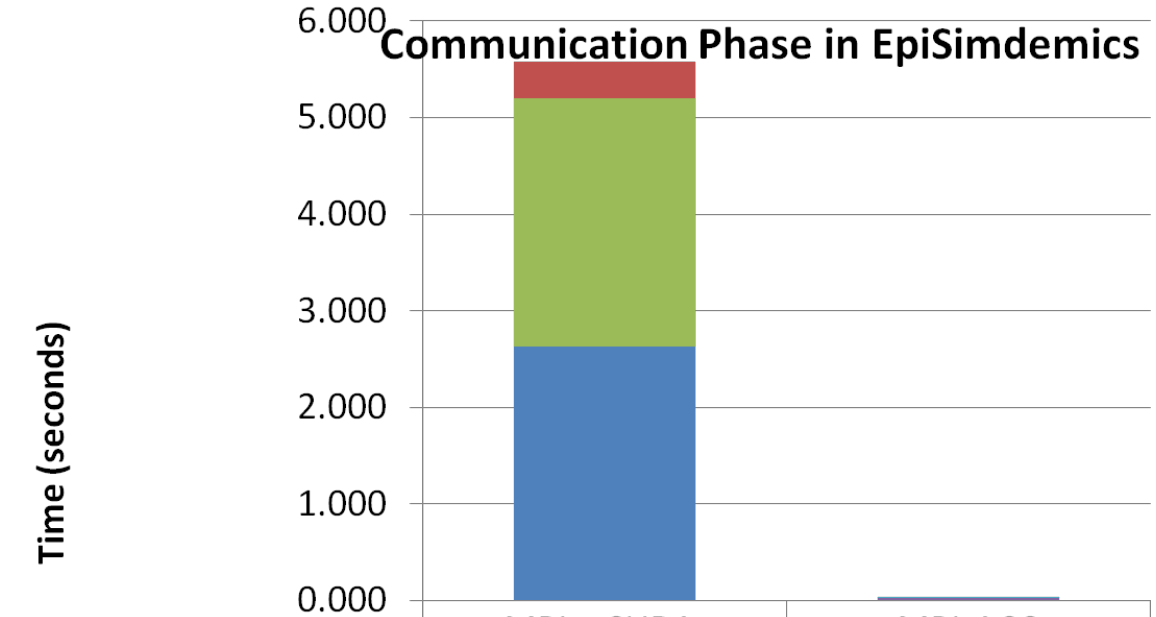


"Enabling Fast, Noncontiguous GPU Data Movement in Hybrid MPI+GPU Environments", John Jenkins, James S. Dinan, Pavan Balaji, Nagiza F. Samatova and Rajeev S. Thakur. IEEE International Conference on Cluster Computing (Cluster), 2012

Case Study for MPI-ACC: Epidemiology



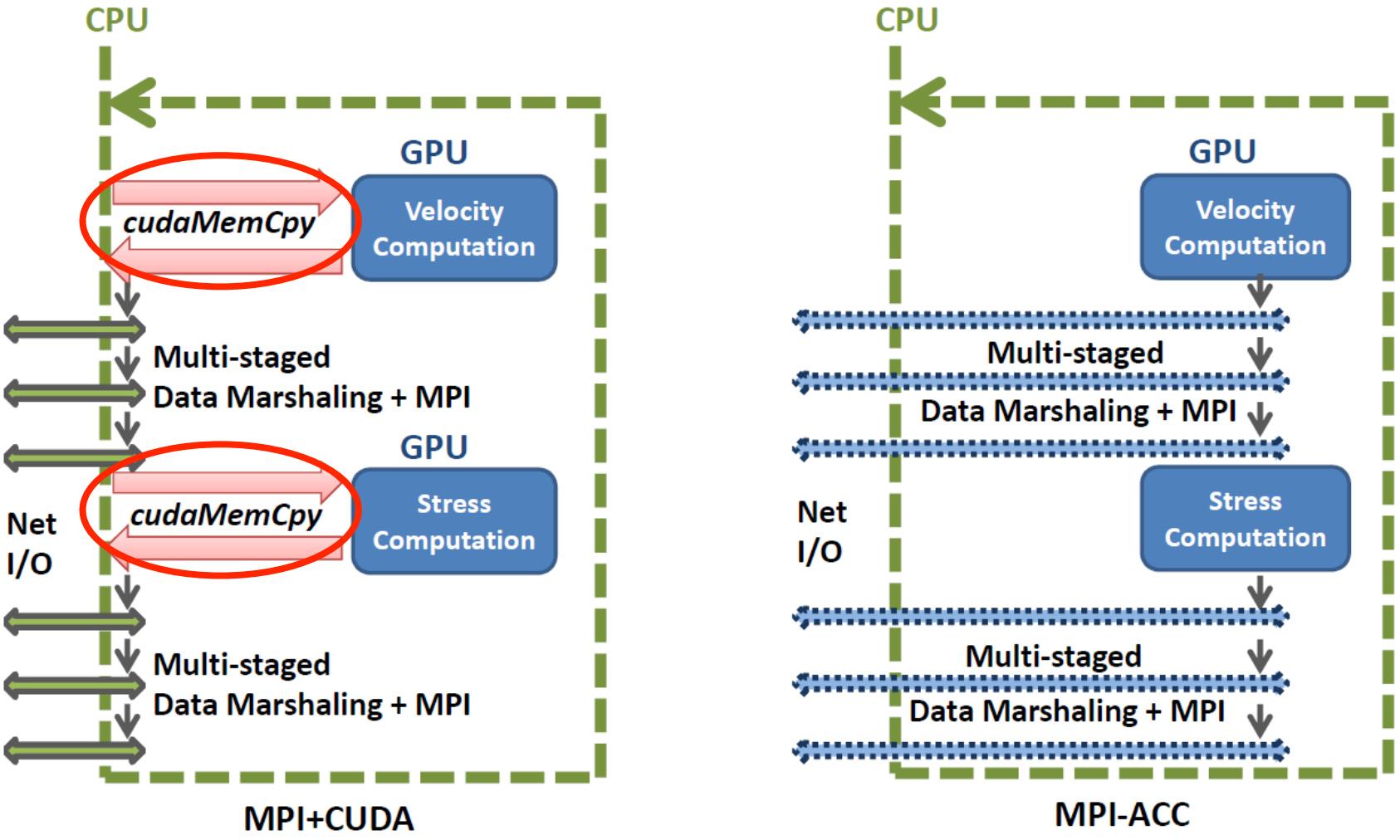
Evaluating the Epidemiology Simulation with MPI-ACC



	MPI + CUDA	MPI-ACC
D-D Copy (Packing)	0	0.003
GPU Receive Buffer Init	0	0.024
H-D Copy	0.382	0
H-H Copy (Packing)	2.570	0
CPU Receive Buffer Init	2.627	0

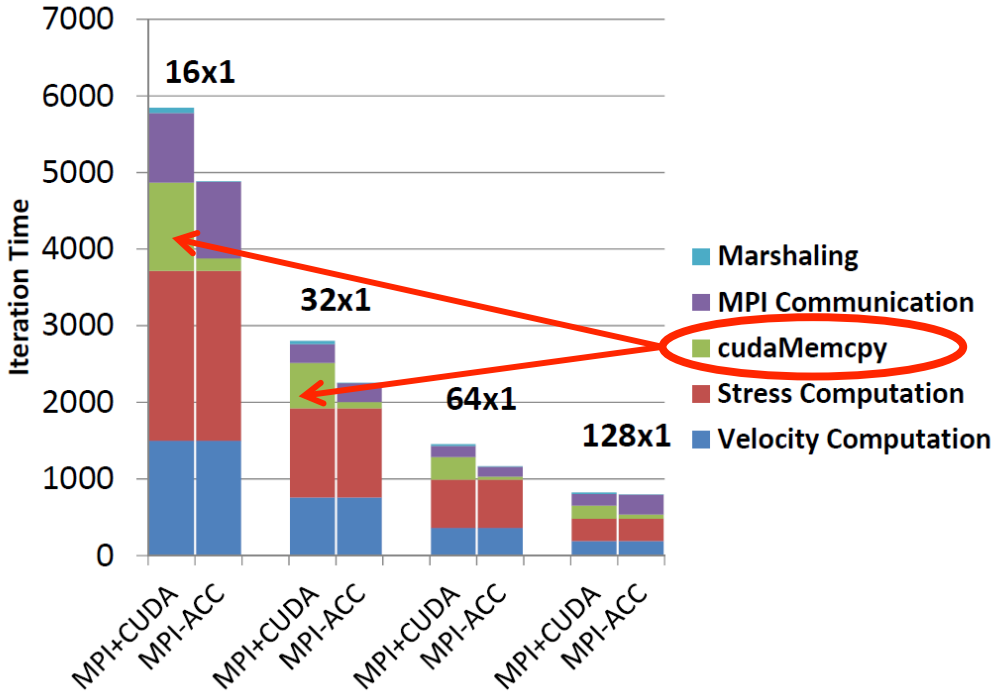
- GPU has *two* orders of magnitude faster memory
- MPI-ACC *enables* new application-level optimizations

Case Study for MPI-ACC: Seismology



Case Study for MPI-ACC: Seismology

- Up to 43% perf. improvement
- Trade-offs
  - Data marshaling on CPU vs. GPU?
    - GPU is better + cudaMemcpy is avoided
  - Data communication from CPU vs. GPU?
    - CPU is better because PCIe hop is avoided



"On the Efficacy of GPU-Integrated MPI for Scientific Applications", Ashwin M. Aji, Lokendra S. Panwar, Feng Ji, Milind Chabbi, Karthik Murthy, Pavan Balaji, Keith R. Bisset, James Dinan, Wu-chun Feng, John Mellor-Crummey, Xiaosong Ma, and Rajeev Thakur. Under Review at the IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2013

Concluding Remarks

- Heterogeneous Memory Systems are becoming common
- Explicit data movement for each segment is infeasible
  - Not productive and impacts performance/power usage
- Unification and abstraction will be key for large heterogeneous systems
- We are working on an example model with MPI
  - The final model need not be MPI

Argonne NATIONAL LABORATORY

Thank You!

Team:

Pavan Balaji, Computer Scientist and Group Lead

James Dinan, Postdoctoral Researcher

David Goodell, Software Development Specialist

Darius Buntinas, Software Development Specialist

Rajeev Thakur, Deputy Division Director