

# Challenges in Scaling MPI to Exascale

**Rajeev Thakur**, Pavan Balaji, Darius Buntinas, Jim Dinan,  
Dave Goodell, Bill Gropp,\* Rusty Lusk, Rob Ross, Marc Snir

Argonne National Laboratory

\*University of Illinois

# Outline

- Role of MPI and MPICH for Exascale
- Recent Research Areas
  - Scalability for very large-scale systems
  - Managing and dealing with complex hardware (processors, networks)
  - Multi-model programming
- Ongoing New Research Directions
  - MPI-3
  - Extensions to MPI
    - Active messages, dynamic tasks, compiler support for MPI
    - Composable programming models
    - Integrating MPI and Accelerators



# MPI on the Largest Machines Today

- Systems with the largest core counts in November 2011 Top500 list (excluding GPU cores)

RIKEN K computer	705,024 cores
Jülich BG/P	294,912 cores
Oak Ridge Cray XT5	224,162 cores
LLNL BG/L	212,992 cores
Argonne BG/P	163,840 cores
NERSC Cray XE6	153,408 cores

- Within the next few months, we will have systems with more than a million cores
- For example, the Sequoia machine at Livermore will be an IBM BG/Q with 1,572,864 cores (~1.6 million cores)

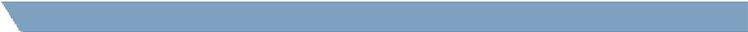


# Potential System Architecture Targets

System attributes	2010	"2015"		"2018"	
System peak	2 Peta	200 Petaflop/sec		1 Exaflop/sec	
Power	6 MW	15 MW		20 MW	
System memory	0.3 PB	5 PB		32-64 PB	
Node performance	125 GF	0.5 TF	7 TF	1 TF	10 TF
Node memory BW	25 GB/s	0.1TB/sec	1 TB/sec	0.4TB/sec	4 TB/sec
Node concurrency	12	O(100)	O(1,000)	O(1,000)	O(10,000)
System size (nodes)	18,700	50,000	5,000	1,000,000	100,000
Total Node Interconnect BW	1.5 GB/s	20 GB/sec		200GB/sec	
MTTI	days	O(1day)		O(1 day)	

Source: Andy White and Rick Stevens talk on "A decadal DOE plan for providing exascale applications and technologies for DOE mission needs," DOE ASCAC meeting, March 2010





# Role of MPI and MPICH for Exascale



# MPICH: A High Performance Implementation of MPI

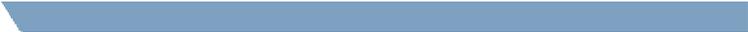
- MPICH is our research vehicle for doing research in MPI as well as other runtime capabilities that we expect to see in large-scale systems
- Project Goals:
  - Be the MPI implementation of choice for the highest-end parallel machines
    - Many of the largest machines in the Top500 list use MPICH2-based implementations
  - Carry out the research and development needed to scale MPI to exascale
    - Optimizations to reduce memory consumption
    - Fault tolerance
    - Efficient multithreaded support for hybrid programming
    - Performance scalability
    - Extensions to MPI
  - Work with the MPI Forum on standardization and early prototyping of new features



# MPICH Project History

- Conceived in November 1992 (SC meeting when MPI-1 work began)
  - Actual coding started in March 1993
- When MPI-2 was being formulated, we rewrote MPICH to start a second version of the code base called “MPICH2”
  - Current research effort
  - We use “MPICH” and “MPICH2” interchangeably; typically “MPICH” refers to the overall project and “MPICH2” refers to the code base
- MPICH has prototyped various proposals in MPI as they were being proposed
- Has been the first implementation of every released MPI standard
  - MPI-1.0, 1.1, 1.2, 1.3, 2.0, 2.1, 2.2
  - Working on being the first implementation to support MPI-3





# MPICH collaboration with vendors

- Enable vendors to provide high-performance MPI implementations on the leading machines of the future
- Collaboration with IBM on MPI for the Blue Gene/Q
  - Multithreaded optimizations for high concurrent message rates (recent publications in Cluster 2010 and EuroMPI 2010)
  - Memory optimizations
- Collaboration with Cray for MPI on their new interconnect (Gemini)
- Continued collaboration with Intel, Microsoft, Myricom, and Ohio State (MVAPICH)

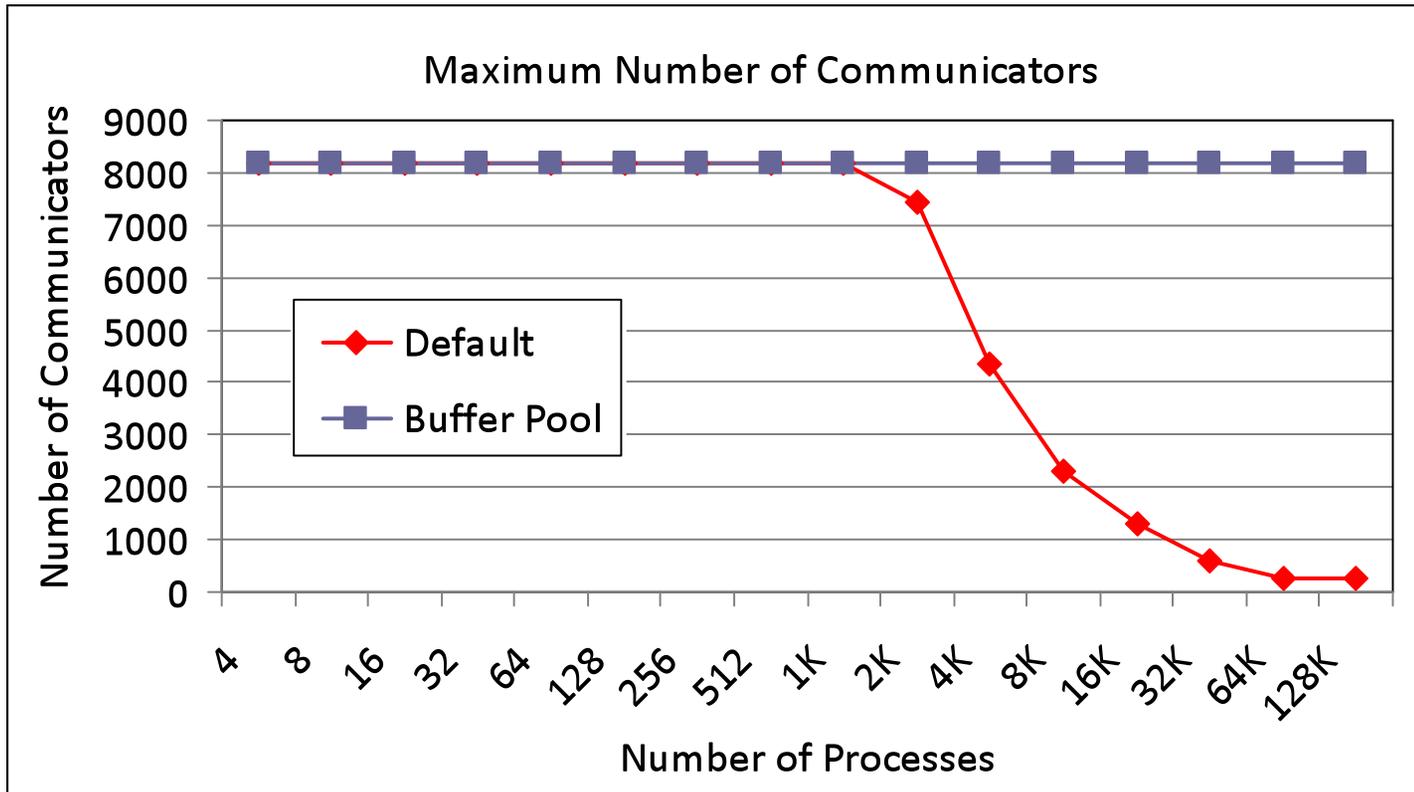


# Essential Factors for Scaling MPI to Exascale

- "MPI on Millions of Cores"
  - Balaji, Buntinas, Goodell, Gropp, Hoefler, Kumar, Lusk, Thakur, Träff, *Parallel Processing Letters*, 21(1):45--60, March 2011
- Performance scalability
  - Of all functions, not just the commonly optimized ones
- Memory consumption scalability
  - Should not grow linearly or worse with the number of processes
- Fault tolerance/resilience
- Efficient support for hybrid programming and composability with other programming models
- Above features need to be supported both in the MPI specification and in MPI implementations
  - Being addressed in the MPI-3 standard
  - Being addressed by MPI implementations (e.g., MPICH)



## Example: Communicator Memory Consumption Fixed



- NEK5000 code failed on BG/P at large scale because MPI ran out of communicator memory. We fixed the problem by using a fixed buffer pool within MPI and provided a patch to IBM.



## Example: MPI\_Comm\_split Performance Scalability Problem Fixed

- A user on our BG/P reported that as he doubled the number of processes, the time taken by MPI\_Comm\_split quadrupled!
- Problem was narrowed down to the algorithm used to implement a stable sort
- Switching to a better local stable sorting algorithm fixed the problem

	OLD	NEW
16,384 procs	1.5 sec	0.105 sec
32,768 procs	6.3 sec	0.126 sec
65,536 procs	25.3 sec	0.168 sec
131,072 procs	101.2 sec	0.255 sec



# Scalable Memory Usage for Internal Data Structures

- “Scalable Memory Use in MPI: A Case Study with MPICH2”
  - Dave Goodell, Bill Gropp, Xin Zhao, Rajeev Thakur, EuroMPI 2011
- Discussion of areas in MPI that appear to need  $O(p)$  memory on each process and how it could be avoided
  - Group representation
  - Connections and message buffers
  - RMA windows
  - Nonscalable arguments to MPI functions
- Audit of memory usage in MPICH2
- Designed and implemented a fix for one area of nonscalable memory usage – the virtual connection table



## Scalable Memory Usage (contd.)

- In an MPI implementation, each process must maintain at least some state for each other process with which it communicates
- In MPICH2, it is in the virtual connection (VC) object, and there is an  $O(p)$  size VC table allocated in MPI\_Init for easy access
- Rearchitected a core piece of MPICH2 for cleaner abstraction and to enable memory savings through compression and lazy allocation
- In the new scheme, VCs are created lazily only as needed and stored in a hash table
- Result: Memory consumption scales well for common communication patterns
- Current implementation has some performance overhead that we are tracking down



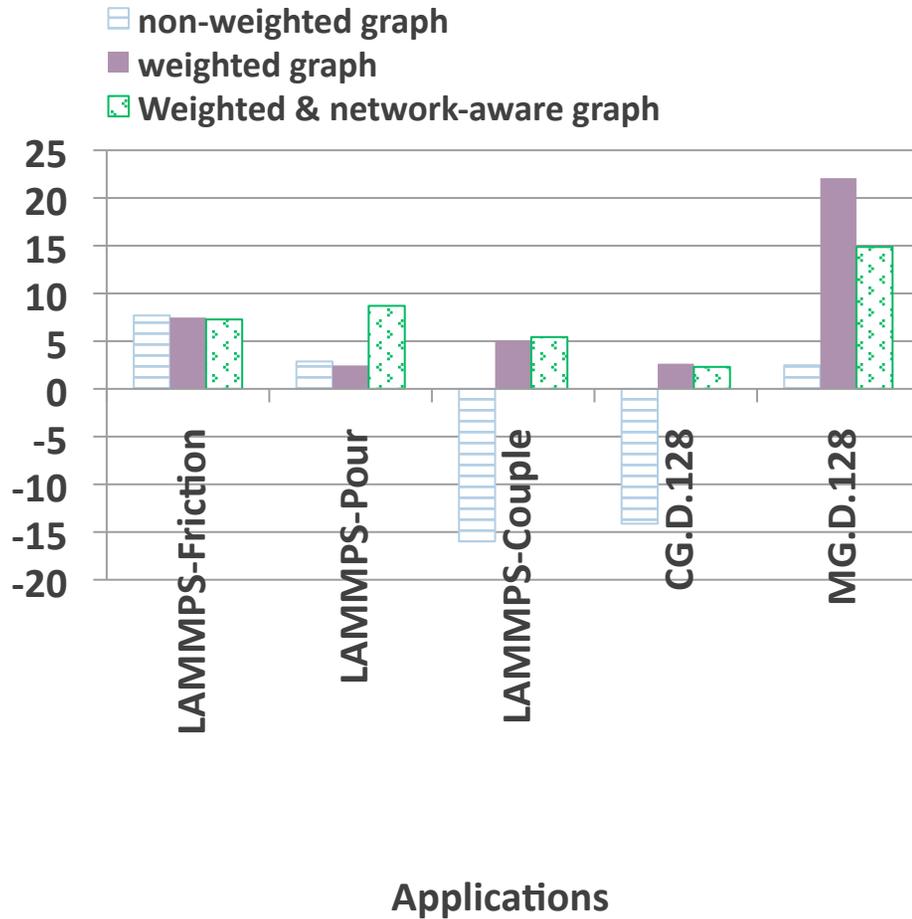
# Dealing with Complex Hardware Topologies

- *“Multi-core and Network Aware MPI Topology Functions”*
  - Mohammad Rashti, Jonathan Green, PavanBalaji, Ahmad Afsahi, Bill Gropp, EuroMPI 2011
- Hardware topologies are becoming increasingly complex
  - Hierarchy in processors (sockets, dies, cores, threads)
  - Scalable networks (torus topologies, non-fully-connected networks)
  - MPI internally does some optimizations to maximize performance for various architectures, but often times this is not enough – we need help from applications to tell us how they are going to communicate
- Topology-aware mapping of MPI ranks to cores is critical for performance at extreme scale
  - MPI has virtual topology functions, but they are often unoptimized
  - This work aims at optimizing the mapping of graph topologies to physical processors, with the help of available tools
    - HWLOC for extracting node architecture
    - IB subnet manager (ibtracert) for extracting network distances
    - Scotch library for graph mapping

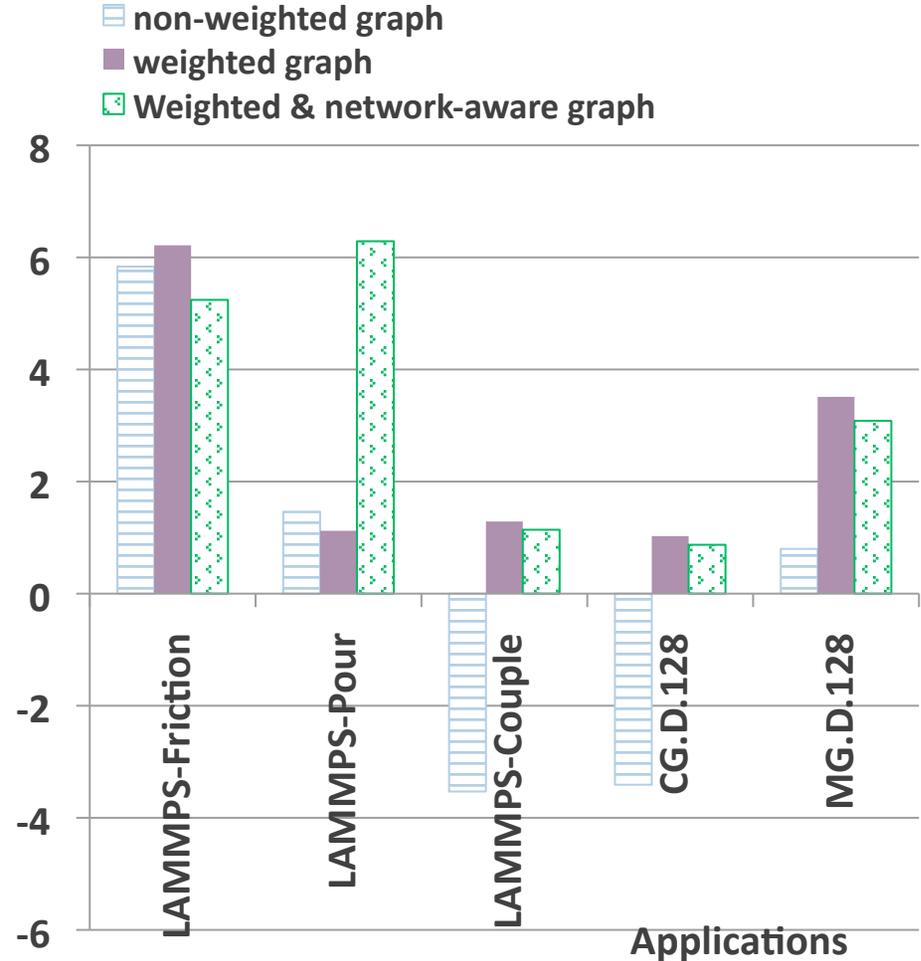


# Applications: Topology-aware Mapping Improvement over Block Mapping (%)

## Communication Time Improvement



## Run-time Improvement



On a 128-core cluster



# Enabling MPI+X Hybrid Programming

- MPI is good at moving data between address spaces
- Within an address space, MPI can interoperate with other “shared memory” programming models
- Useful on future machines that will have limited memory per core
- (MPI + X) Model: MPI across address spaces, X within an address space
- Examples:
  - MPI + OpenMP
  - MPI + UPC/CAF (here UPC/CAF address space could span multiple nodes)
  - MPI + CUDA/OpenCL on GPU-accelerated systems
- Precise thread-safety semantics of MPI enable such hybrid models
- We are exploring further enhancements to MPI to support efficient hybrid programming

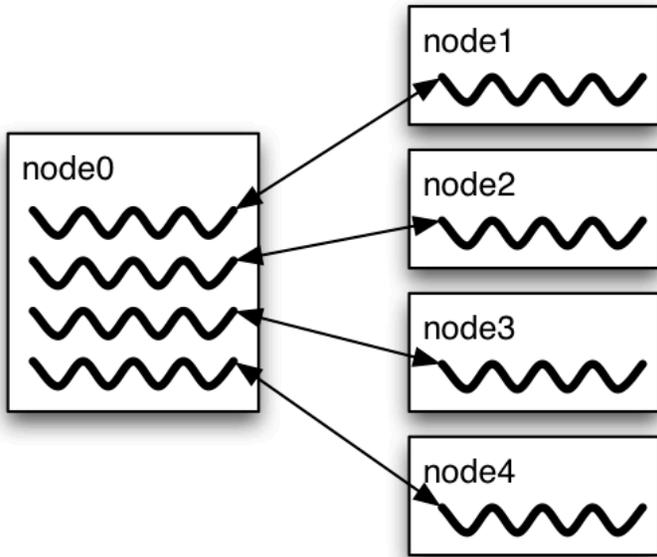


# Efficient Multithreaded Communication for Hybrid Programming

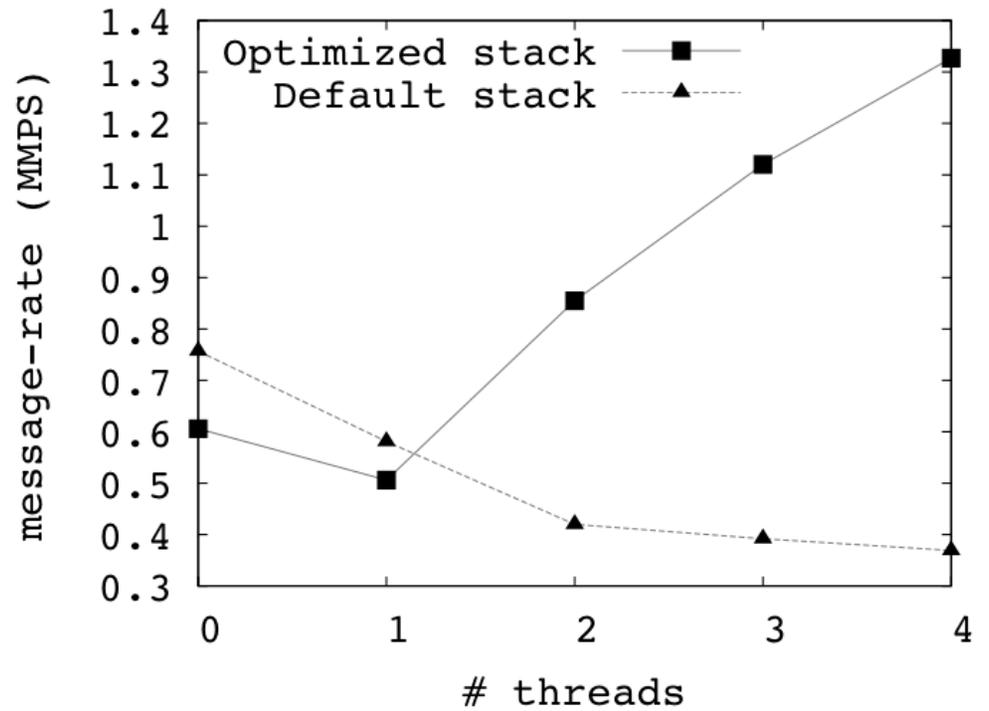
- “Minimizing MPI Resource Contention in Multithreaded Multicore Environments,” Goodell, Balaji, Buntinas, Doza, Gropp, Kumar, de Supinski, Thakur, *Cluster 2010*
  - Reference counting of MPI objects in multithreaded environments requires atomic updates, which are expensive under contention
  - New algorithm proposed that uses a garbage collection method that avoids the need for atomic reference counting
- “Enabling Concurrent Multithreaded MPI Communication on Multicore Petascale Systems,” Doza, Kumar, Balaji, Buntinas, Goodell, Gropp, Ratterman, Thakur, *EuroMPI 2010*
  - Work with IBM on optimizing multithreaded communication
  - Uses a combination of a multichannel-enabled network interface, fine-grained locks, lock-free atomic operations, and message queues specifically designed for concurrent multithreaded access



# Message Rate Results on BG/P

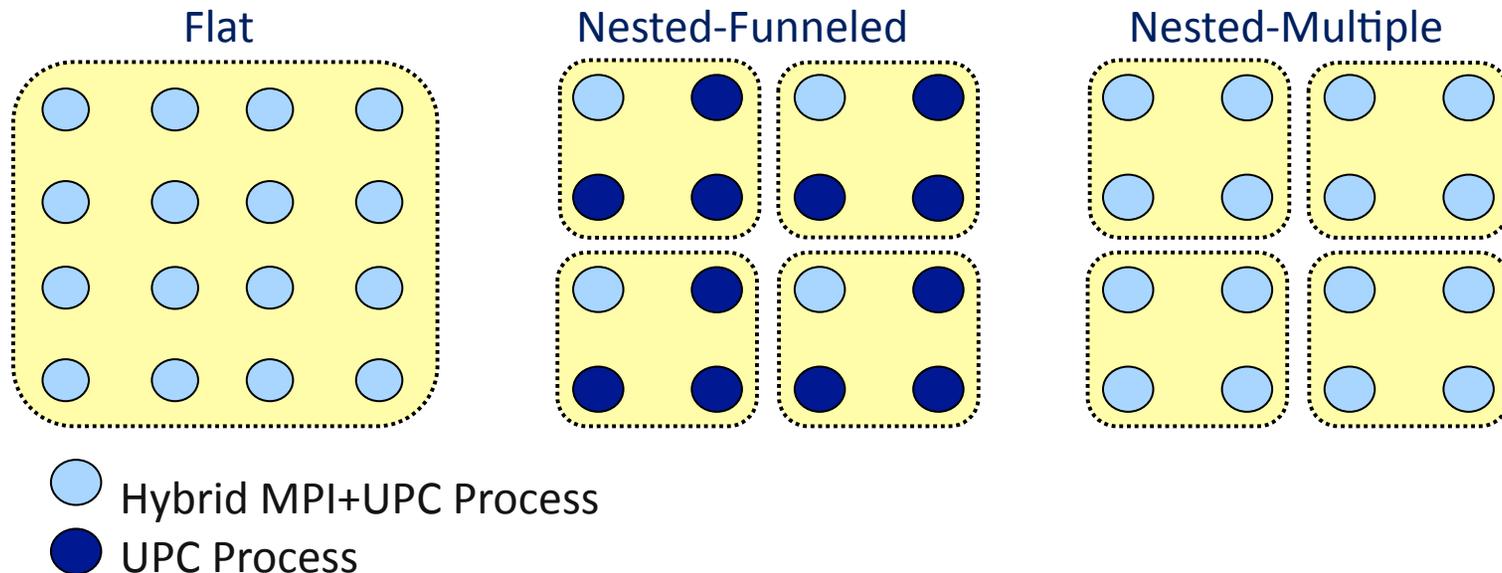


Message Rate Benchmark

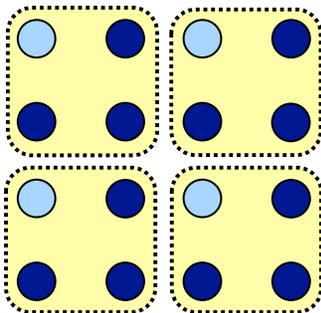
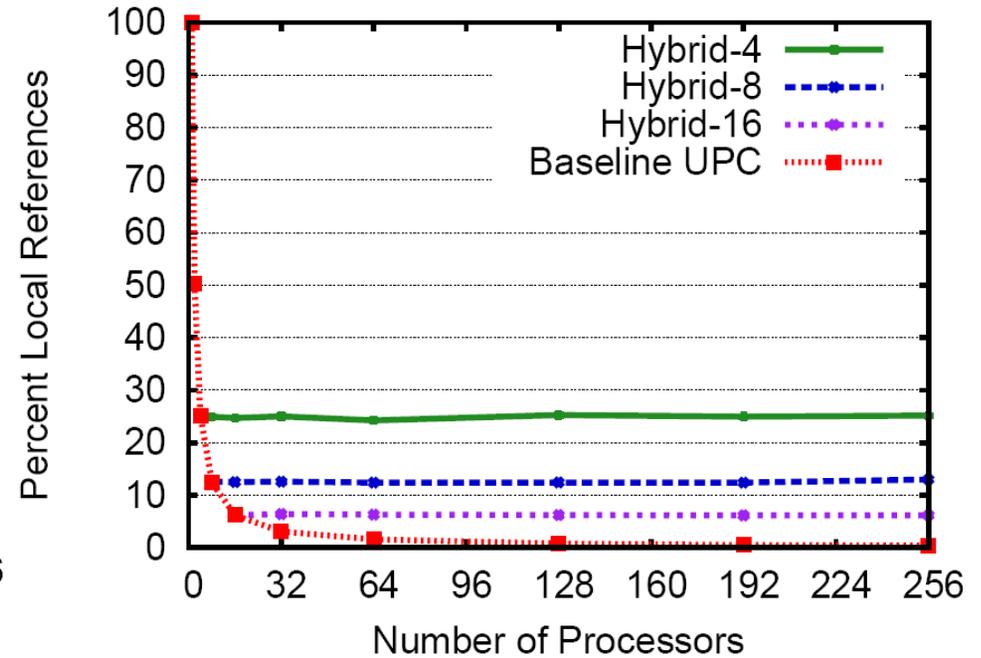
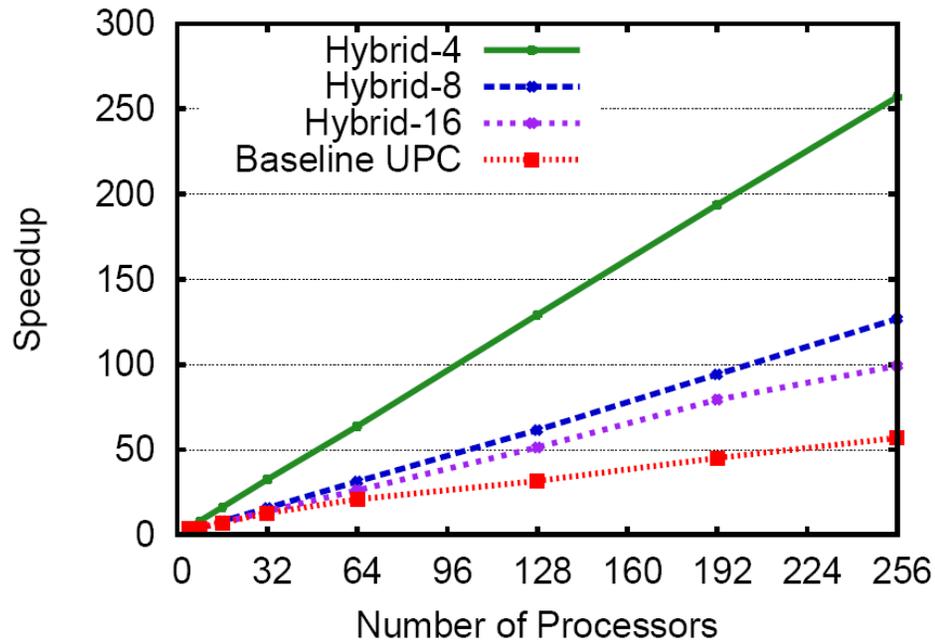


# Multi-Model Programming (MPI+UPC)

- “Hybrid Parallel Programming with MPI and Unified Parallel C” James Dinan, Pavan Balaji, Ewing Lusk, P. Sadayappan, Rajeev Thakur, ACM Conf. on Computing Frontiers, May 2010
- Explores the use of a combination of UPC and MPI in programs
  - UPC within an address space (potentially across multiple nodes)
  - MPI across address spaces
- Multiple models proposed



# Hybrid MPI+UPC Barnes-Hut Algorithm

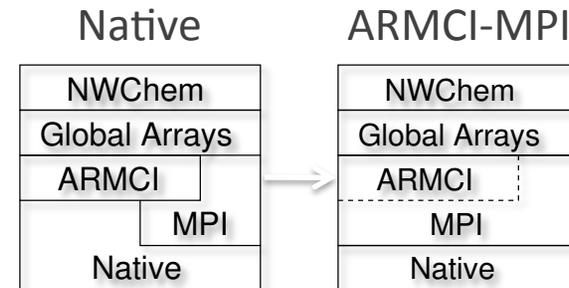


- Nested-funneled model
  - Tree is replicated across UPC groups
- 51 new lines of code (2% increase)
  - Distribute work and collect results



# Global Arrays on MPI

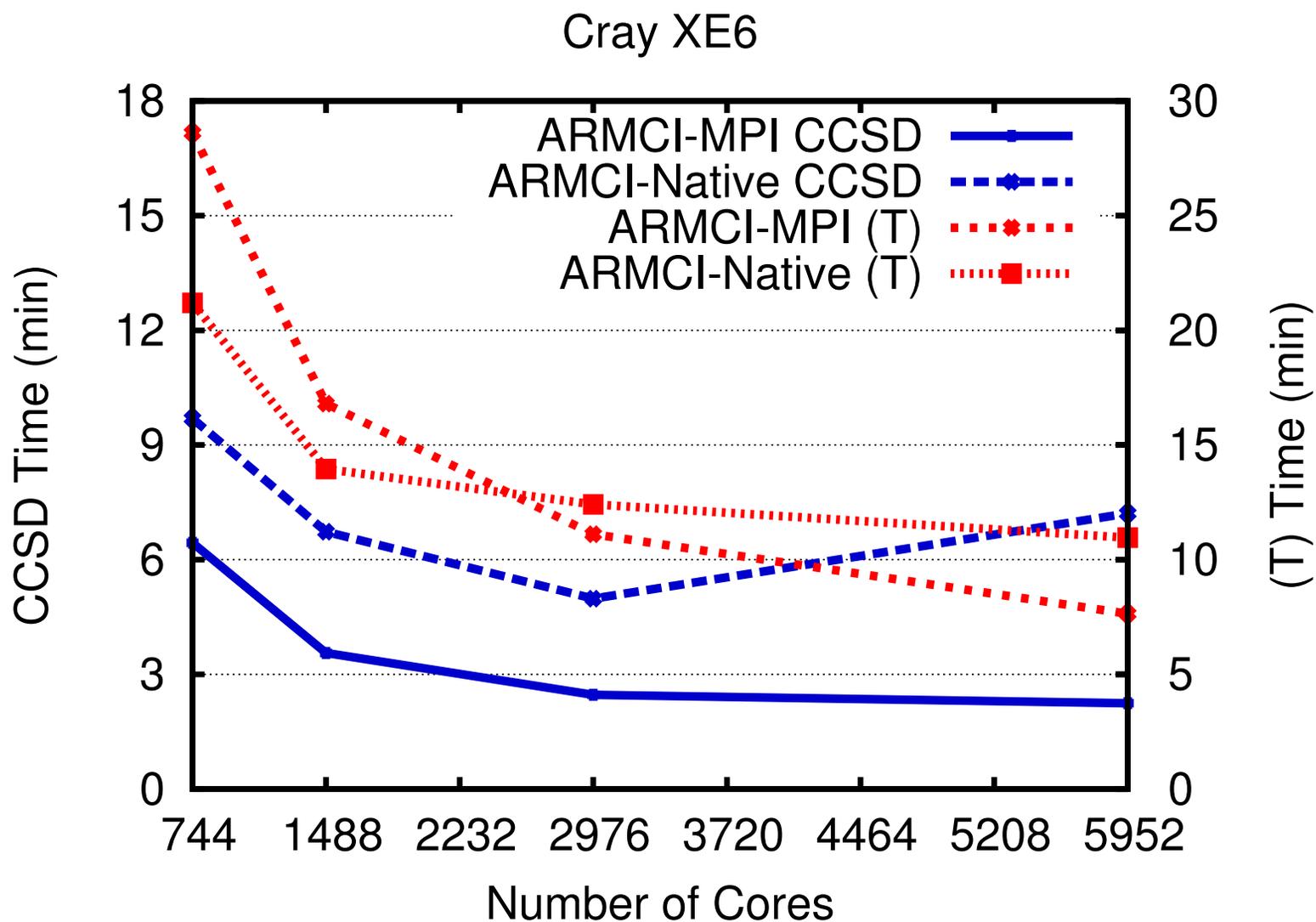
- GA Runtime: ARMCI
  - PGAS runtime system
  - Natively implemented
  - Requires tuning and maintenance
- MPI is ubiquitous
  - MPI has supported one-sided operations for 10 years
  - MPI-RMA is complex, adoption has been slow
- **Goal:** Implement ARMCI on MPI-RMA
  - Portability: GA problems on BG/P, Qlogic IB, IB Torus, new systems
  - Performance: MPI RMA gives access to high performance RDMA
  - Interoperability: More resources available to application
    - Shared progress engine, buffer pinning, network and host resources

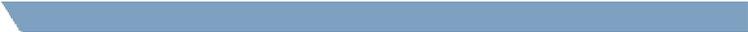


**“Supporting the Global Arrays PGAS Model Using MPI One-Sided Communication,”  
Dinan, Balaji, Hammond, Krishnamoorthy, Tipparaju, IPDPS 2012**



# NWChem Performance (XE6)





# New Research Directions



# MPI-3 and its implementation

- Several people in our group participate actively in the MPI Forum to define MPI-3
- We plan to support MPI-3 in MPICH
- New features expected in MPI-3 include
  - Support for hybrid programming
  - Improved RMA
  - New nonblocking and sparse collectives
  - Fault tolerance
  - New tools interface



# Investigating Extensions to MPI

- Active Messages
  - User-defined callbacks that can be triggered remotely (different progress semantics possible)
  - Critical for high-level models such as Chapel and MADNESS runtime to function correctly over MPI (MADNESS runtime currently wastes a core on each NUMA domain to emulate active messages on its own)
- MPI processes as dynamic processes
  - An MPI process does not need to be a static OS process
    - Making MPI processes more dynamic (as user-level threads) that can switch easily and migrate has several benefits (similar to the Charm++ model)
    - Static MPI processes have their own benefits as well (topology and optimizations are much cleaner and can be done at initialization time)
    - A hybrid model where the user can force some processes to be static and some dynamic might be a good middle ground (MPI-3 endpoints proposal is an example)
- Compiler support for MPI
  - In collaboration with Prof. Qing Yi at UT San Antonio



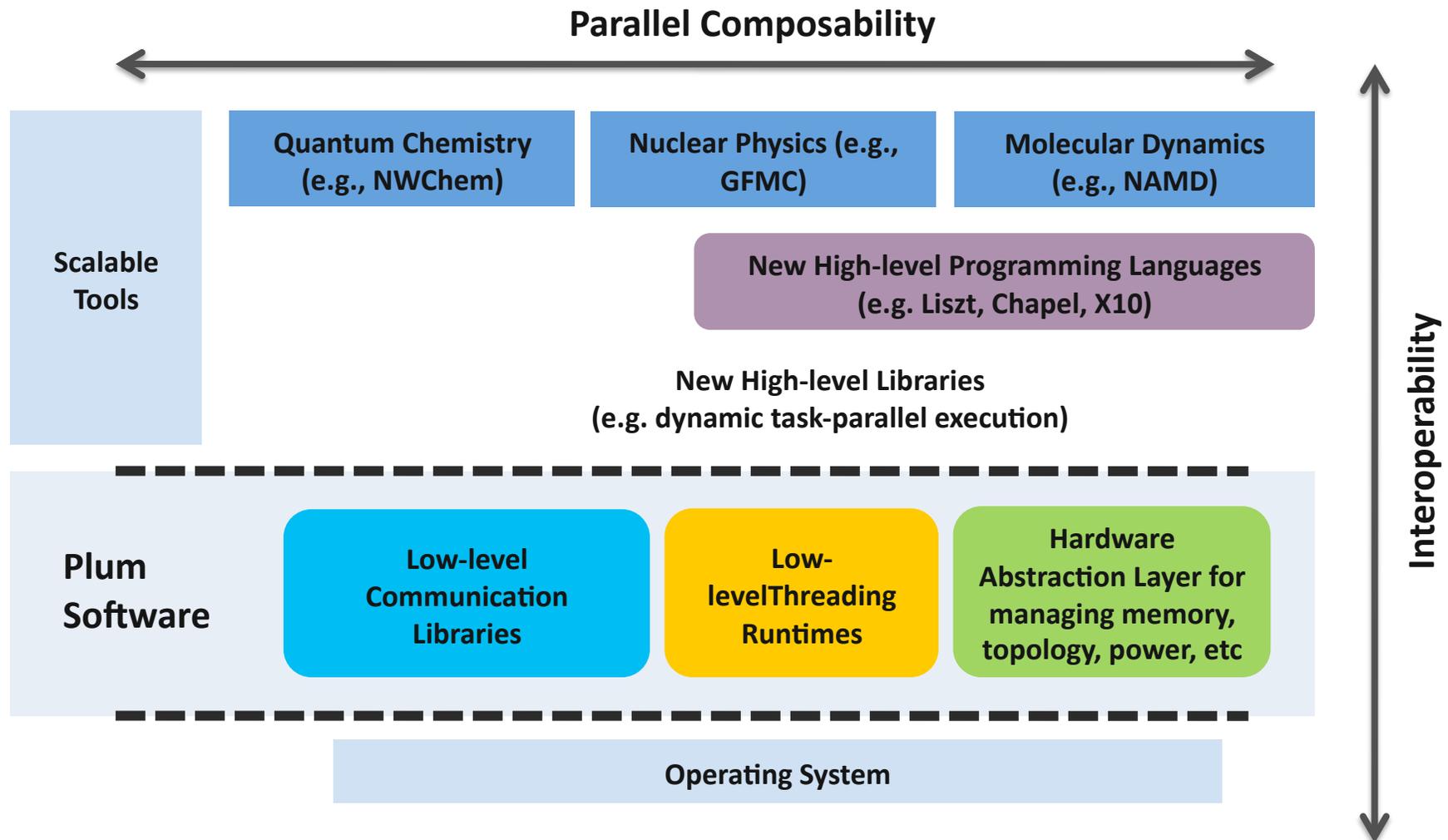
# Interoperability with more models than just threads

- Unfortunately, using multiple programming models is not possible today
  - Programming models are not interoperable today because their runtime systems do not cooperate
  - UPC and CAF use the GASNet runtime system; Global Arrays uses ARMCI; MPI uses its own internal runtime system
  - Impossible to inter-mix these different runtime systems without they knowing of each other
    - Resource conflicts
    - Progress deadlocks
    - Data corruption because of data access contention



# Unistack: Proposed Common Runtime for Multiple Programming Models

Led by Pavan Balaji

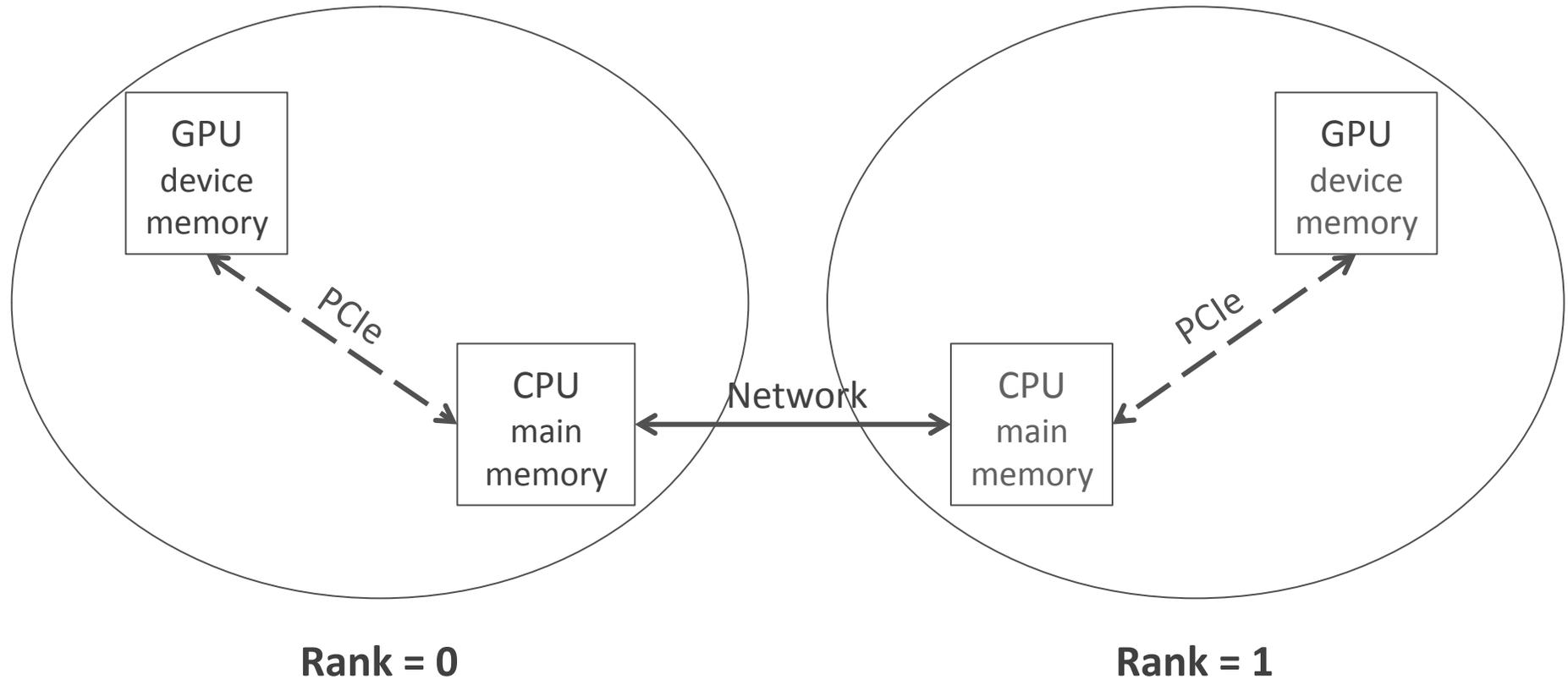


**Goal:** composable and interoperable architecture with replaceable components

**Result:** New, disruptive programming models can be rapidly adopted



# Interoperability with GPUs: Current Data Model



```
if(rank == 0)
{
  cudaMemcpy(s_buf, s_dev_buf, D2H);
  MPI_Send(s_buf, .. ..);
}
```

```
if(rank == 1)
{
  MPI_Recv(r_buf, .. ..);
  cudaMemcpy(r_dev_buf, r_buf, H2D);
}
```

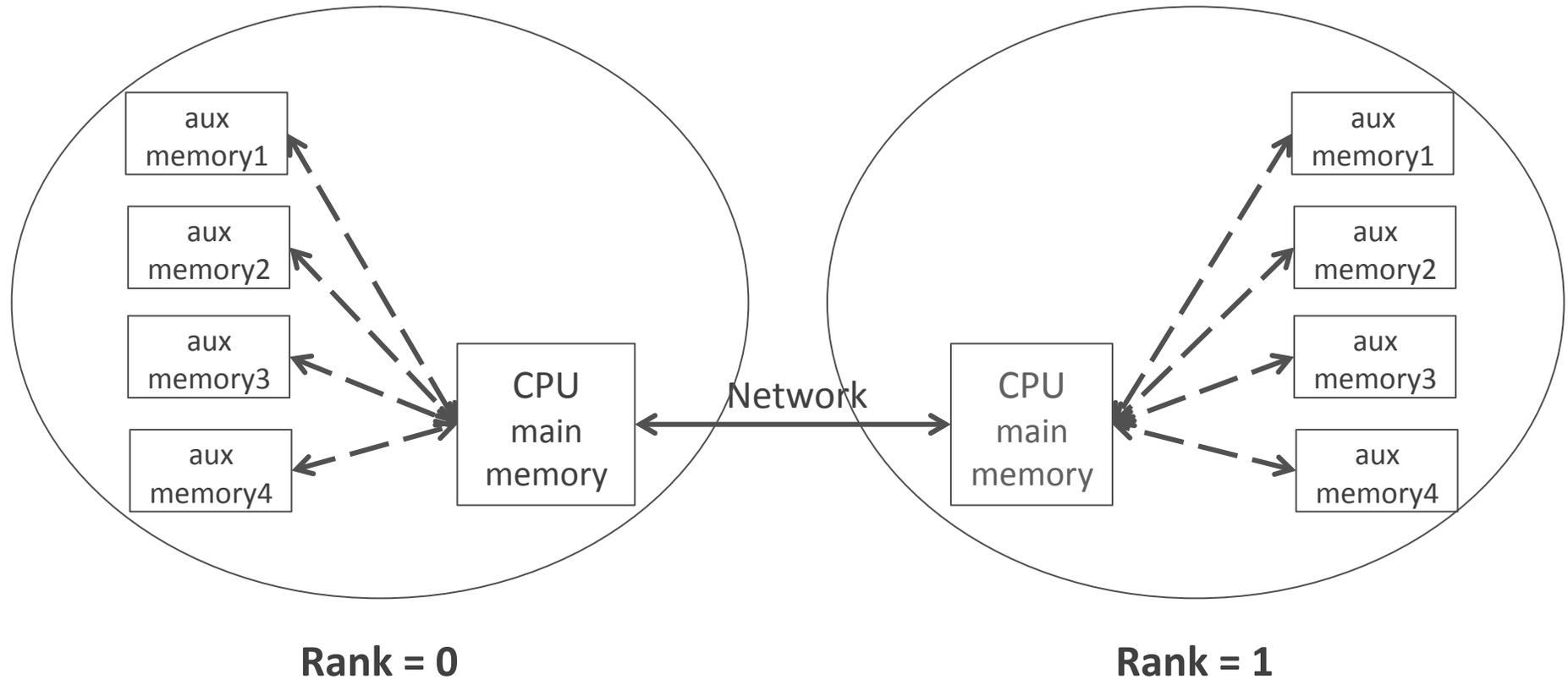


# Project Goals

- Productivity Goal (API)
  - Implement the rich data transfer interface of MPI for CUDA, OpenCL, PGAS models, etc
    - Contiguous data, different data-types (noncontiguous), collectives, one-sided communication
- Performance Goal
  - Pipeline the data movement between GPU memory, host memory and remote node using architecture specific enhancements
    - NVIDIA: GPU Direct
    - Multi-stream copies between GPU and memory (multiple command queues can benefit from parallelism in the DMA engine)
  - Future architectures:
    - Zero-copy data movement if accelerators have direct network access
    - Eliminate “GPU-to-host” data transfers if the heterogeneous processors share memory spaces
- All of the above should happen *automatically* within the MPICH implementation, i.e. applications should not redo their data movement for each architecture



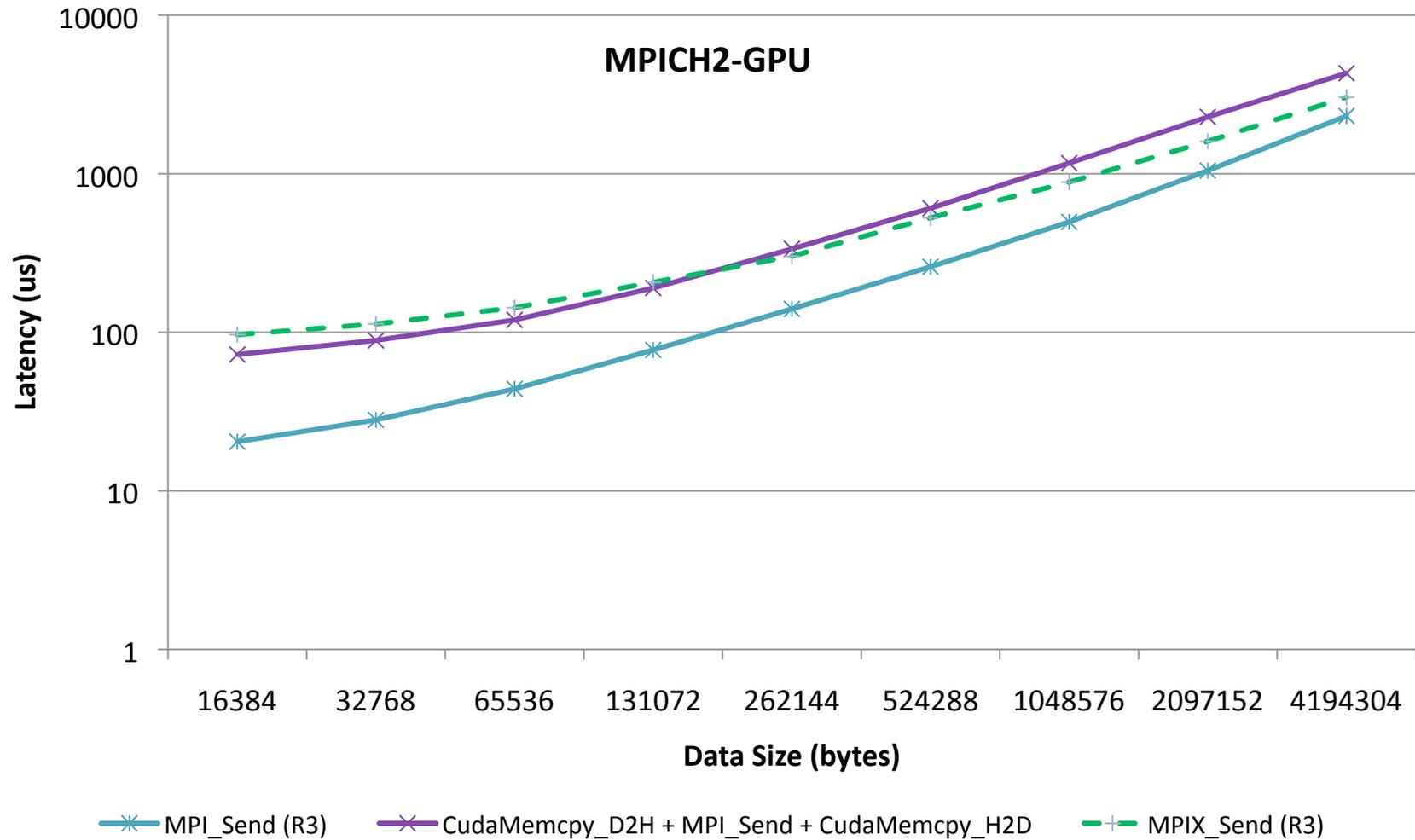
# Interoperability with GPUs: New Data Model



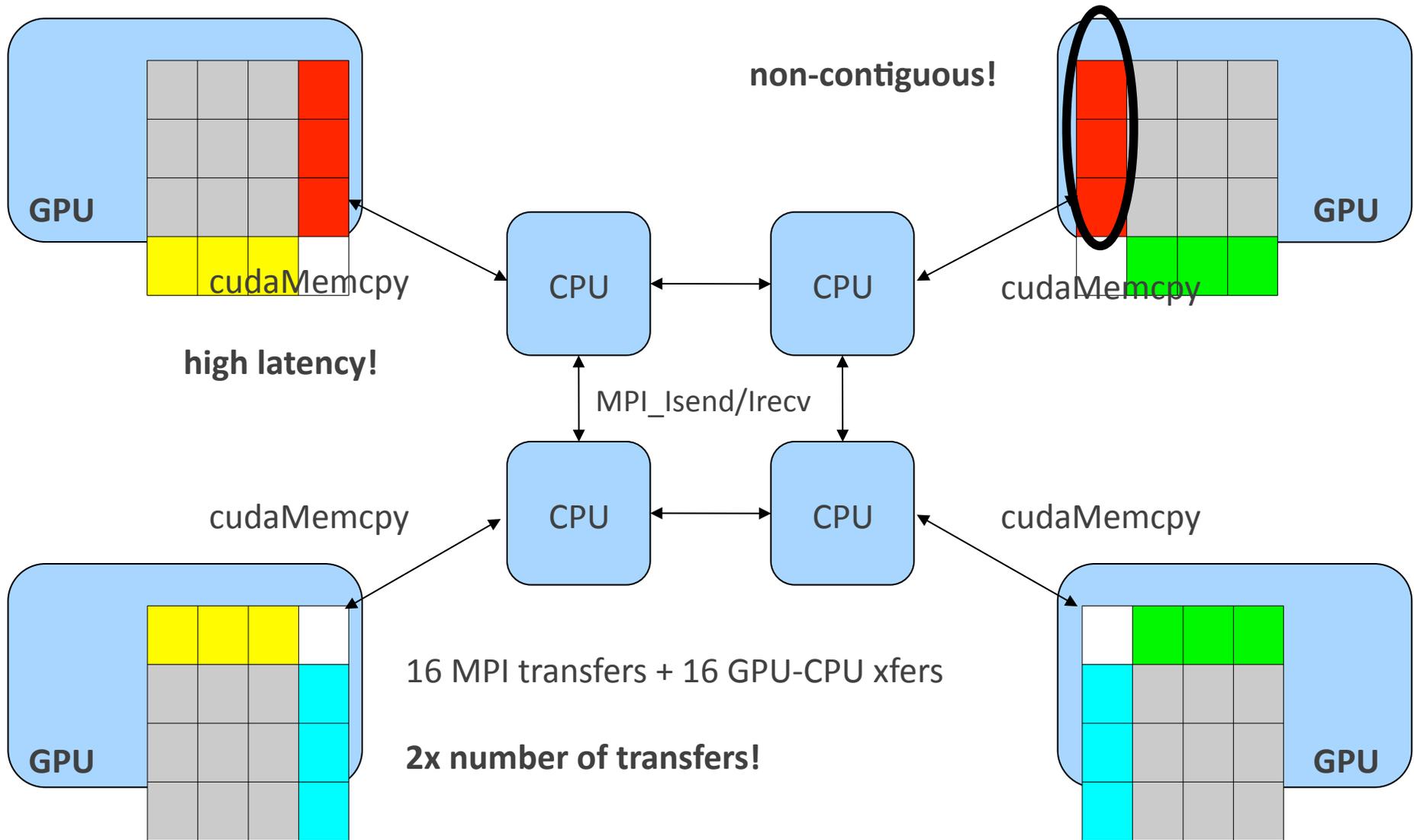
```
if(rank == 0)
{
  MPI_Send(s_aux_buf, .. ..);
}
```

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

# Experimental Results (CUDA - RNDV mode)

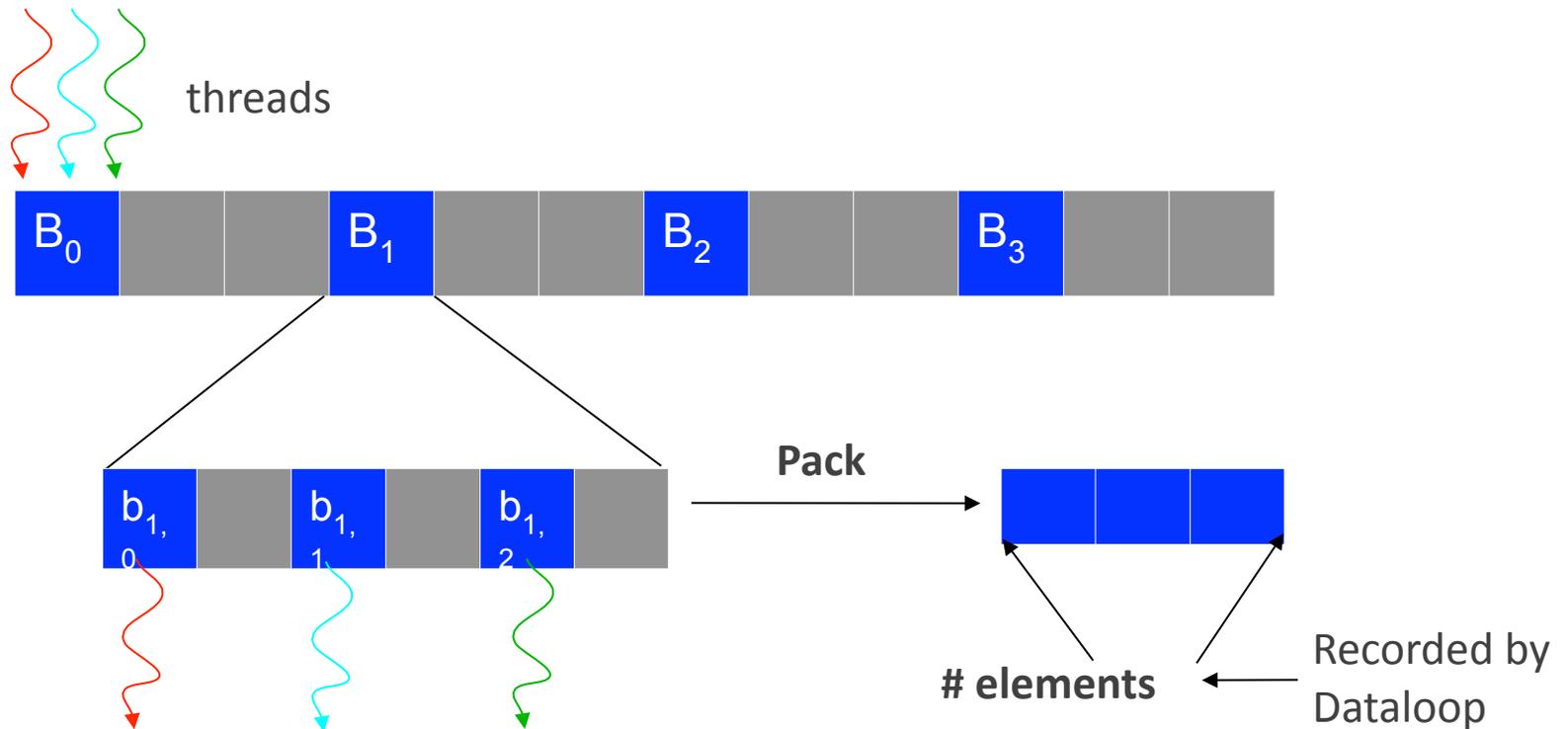


# MPI + GPU Example - Stencil Computation



# GPU optimizations for Data Packing

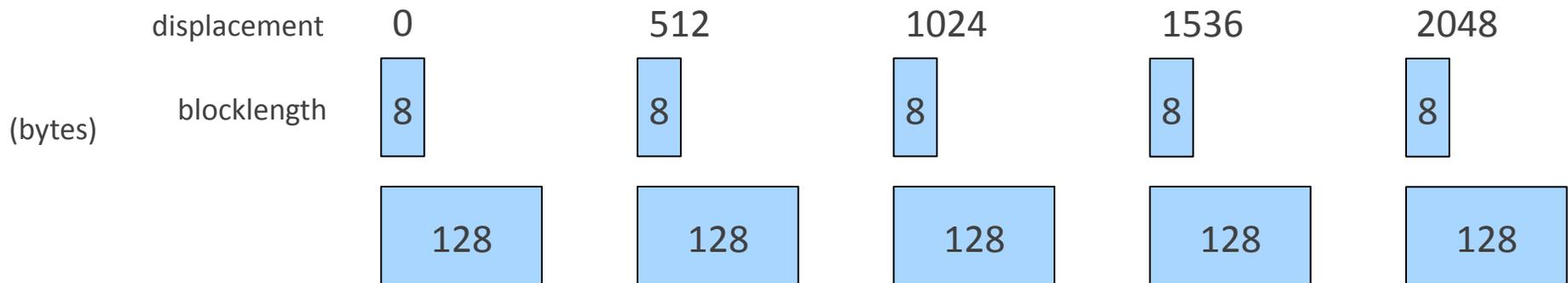
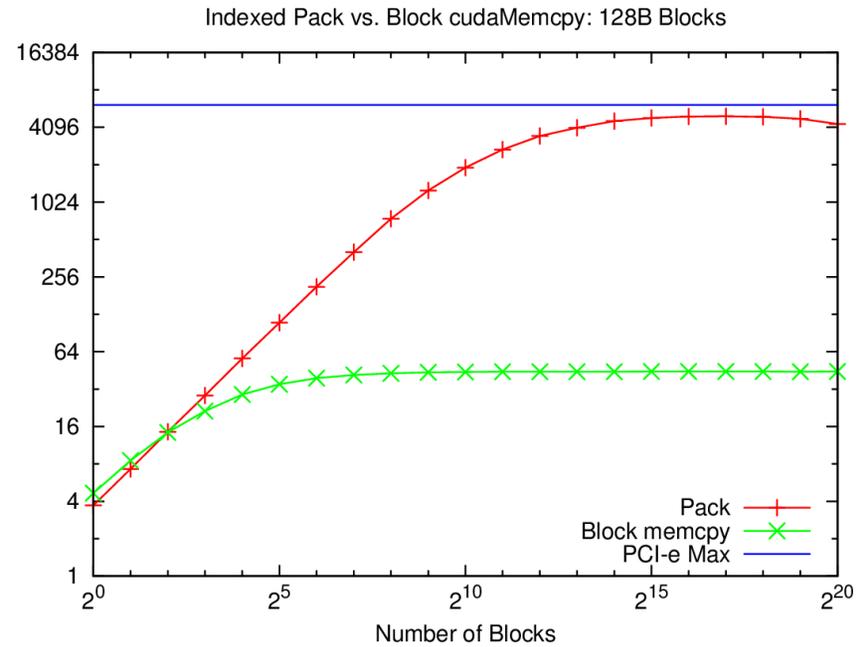
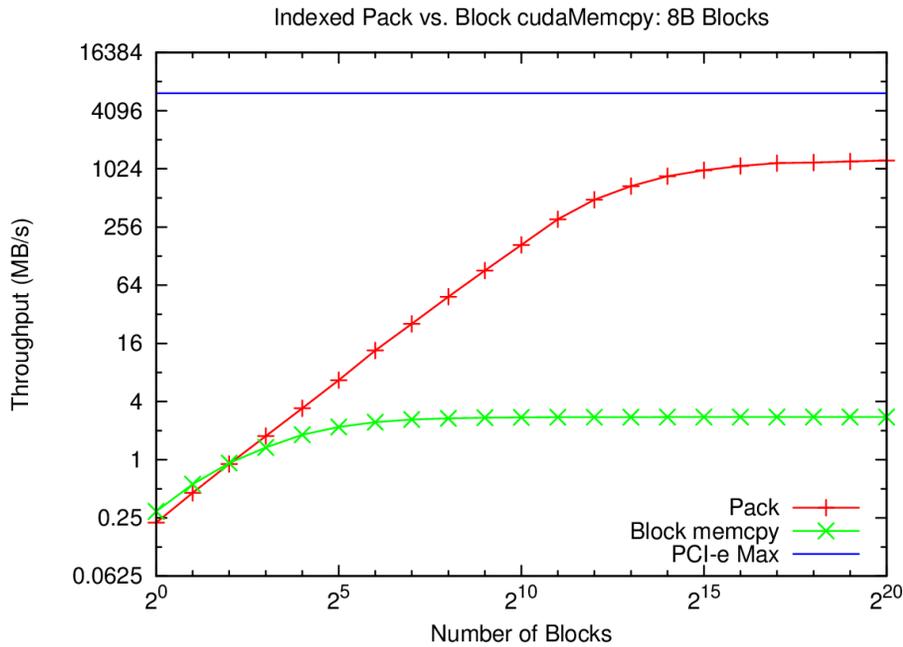
- Element-wise traversal by different threads
- Embarrassingly parallel problem, except for structs, where element sizes are not uniform



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



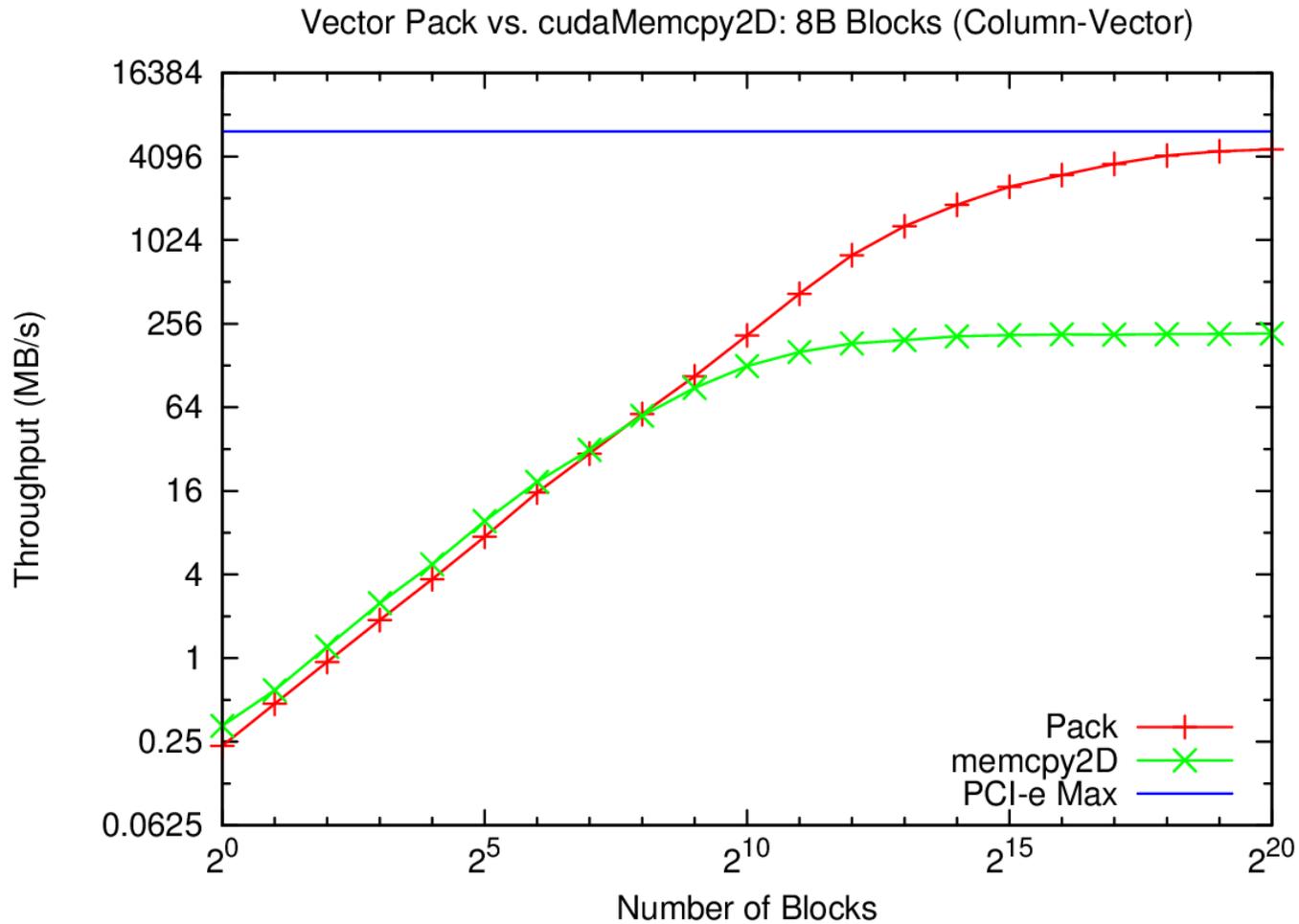
# Packing Throughput (Indexed)



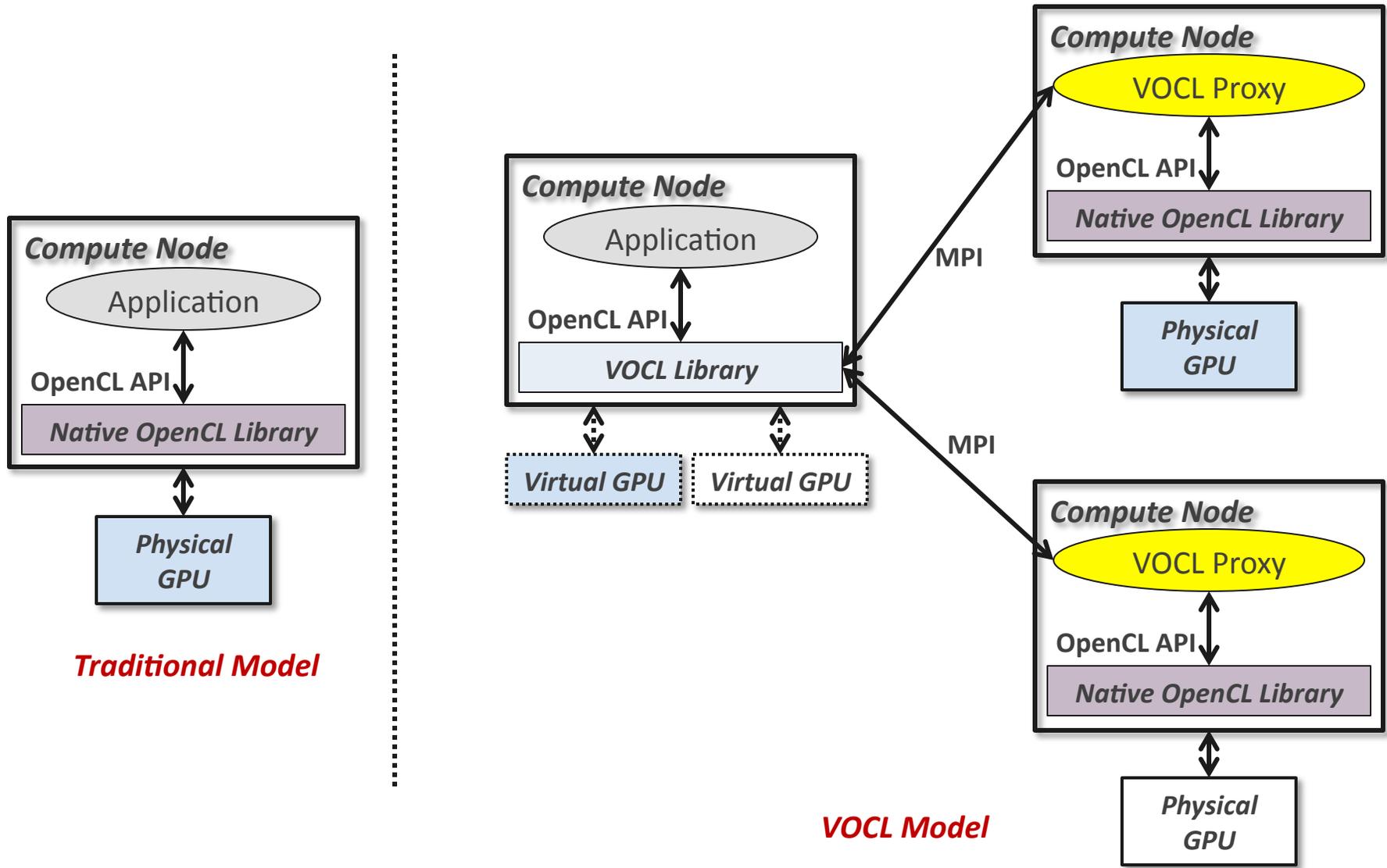
John Jenkins, Accelerating Movement of Non-contiguous Data on Hybrid MPI+GPU Environments, Argonne National Laboratory, 8/15/11



# Packing Throughput (Column-Vector)



# VOCL Framework for Virtualizing GPUs



# MPICH Collaborators/Partners

- Core MPICH developers

- IBM
- INRIA
- Microsoft
- Intel
- University of Illinois
- University of British Columbia



- Derivative implementations

- Cray
- Myricom
- Ohio State University



- Other Collaborators

- Absoft
- Pacific Northwest National Laboratory
- QLogic
- Queen's University, Canada
- Totalview Technologies
- University of Utah



# Summary

- MPI has succeeded because
  - features are orthogonal (complexity is the product of the number of *features*, not routines)
  - complex programs are no harder than easy ones
  - open process for defining MPI led to a solid design
  - programmer can control memory motion and program for locality (critical in high-performance computing)
  - precise thread-safety specification has enabled hybrid programming
- MPI is ready for scaling to extreme scale systems with millions of cores barring a few issues that can be (and are being) fixed by the MPI Forum and by MPI implementations

