Argonne NATIONAL LABORATORY

A One-Sided View of HPC: Global-View Models and Portable Runtime Systems

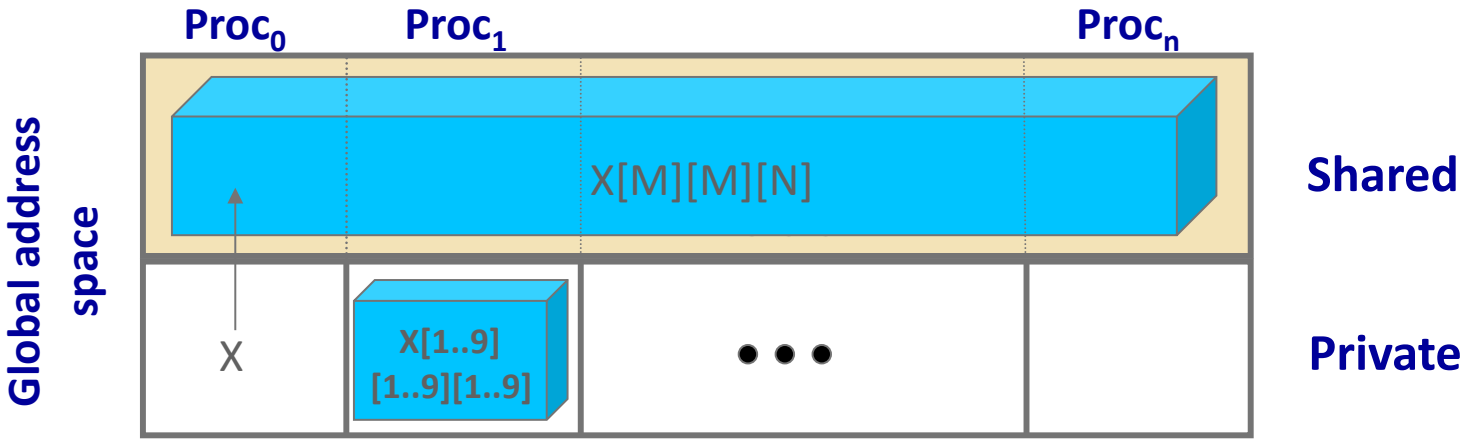
James Dinan

James Wallace Gives Postdoctoral Fellow

Argonne National Laboratory

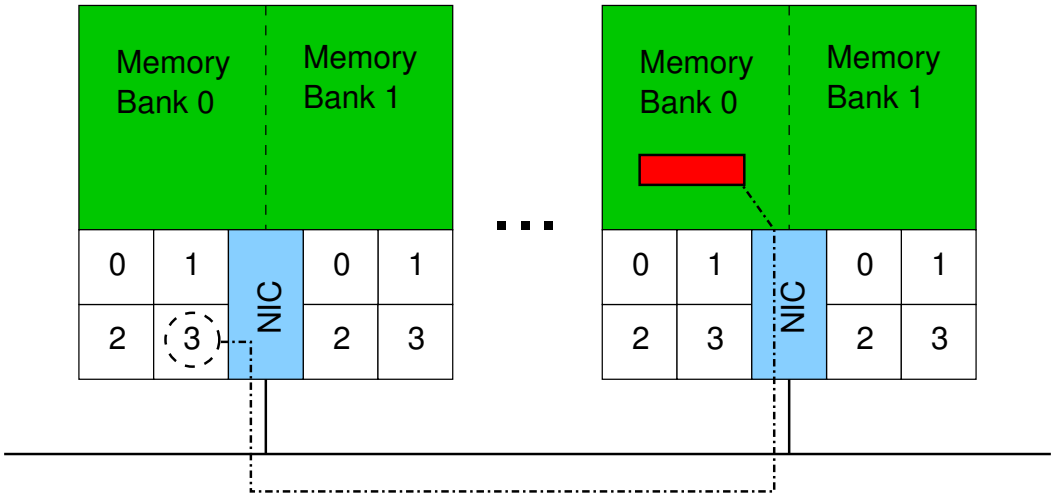
U.S. DEPARTMENT OF ENERGY

Why Global-View?



- Mechanism for dealing with large datasets
  - Similar convenience as shared memory
    - Don't operate on data in-place
    - No coherence across data copies
- Mechanism for coping with sparse, unbalanced computations
  - Work distribution decoupled from data distribution
  - More flexible load balancing
- Computational chemistry apps exhibit both of these characteristics
- Specifically Investigate: NWChem and Global Arrays

Why One-Sided?

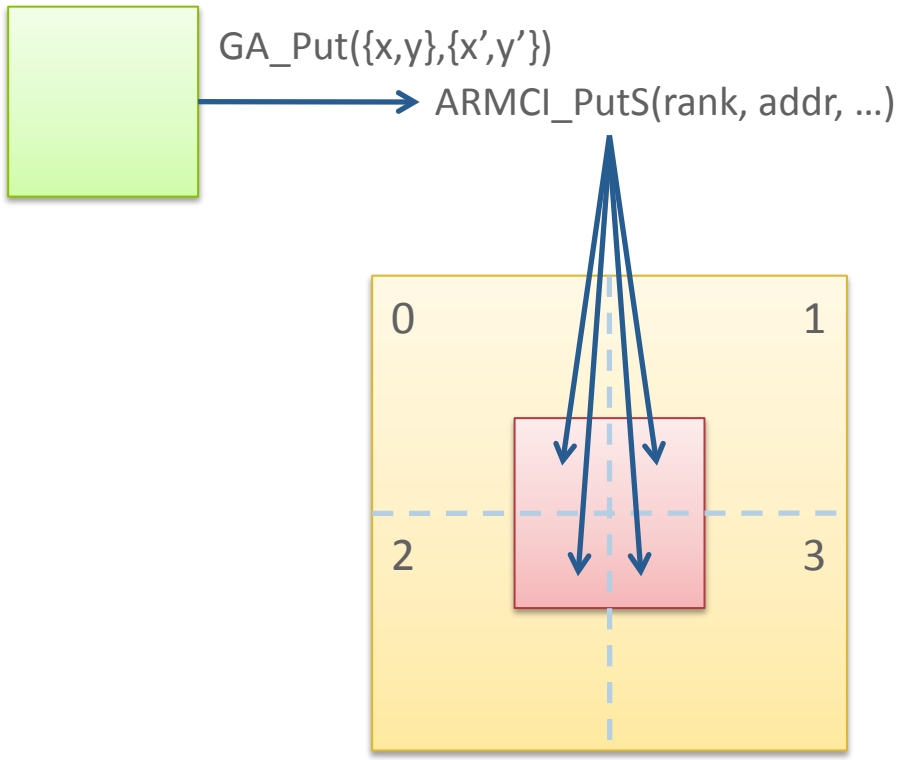


- Explicit message exchange couples two operations:
  1. Synchronization of processes
     - Operation completes when sender calls send and receiver calls receive
  2. Data movement
- One-sided decouples these operations
  - Enables asynchronous data movement
  - Efficient support through RDMA-capable networks

Global Arrays and ARMCI

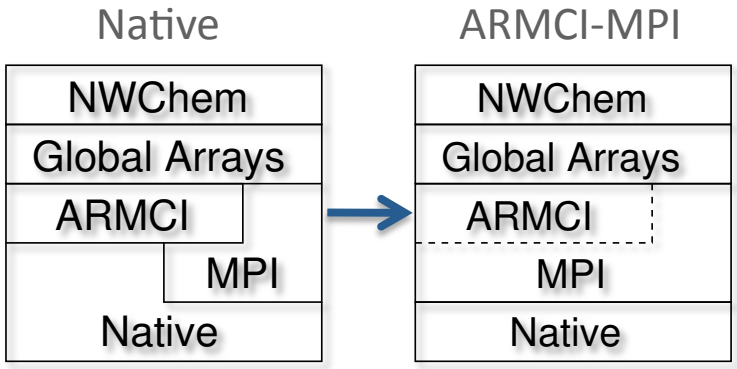
The Aggregate Remote Memory Copy Interface

- Global Arrays (GA)
  - Distributed, shared arrays
- ARMCI
  - GA's runtime system
  - Manages array data
  - Provides portability
- Async., one-sided comm.
  - *Get*, *put*, *accumulate*, …
  - Noncontiguous operations
- Mutexes, atomics, collectives, processor groups, …
- Consistency model
  - Remote: Location consistent
  - Local: Direct load/store allowed



The GA/ARMCI Software Stack

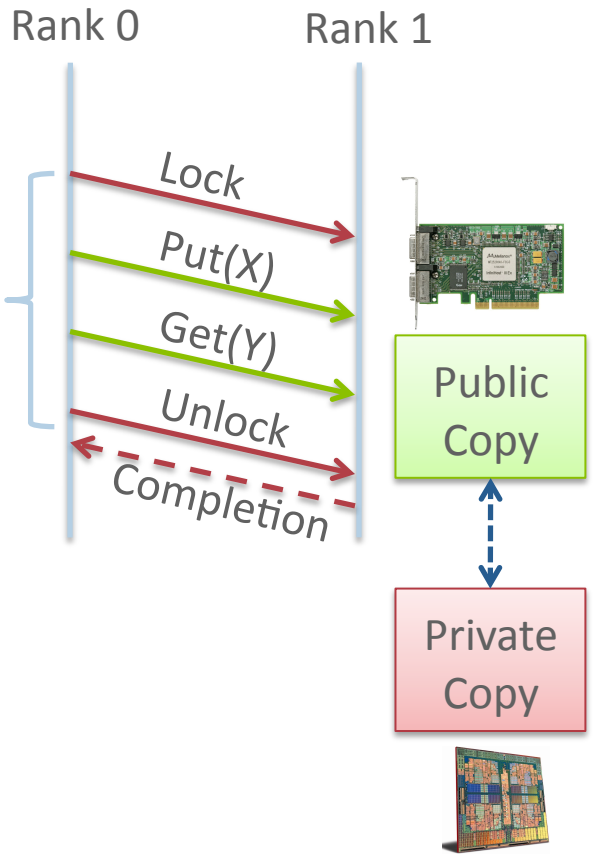
- ARMCI Support
  - Natively implemented
  - Sparse vendor support
  - Implementations lag systems
- MPI is ubiquitous
  - Support one-sided for 15 years



- ***Goal***: Use MPI RMA to implement ARMCI
  1. Portable one-sided communication for NWChem users
  2. MPI-2: drive implementation performance, one-sided tools
  3. MPI-3: motivate features
  4. Interoperability: Increase resources available to application
     - ARMCI/MPI share progress, buffer pinning, network and host resources
- ***Challenge***: Mismatch between MPI-RMA and ARMCI

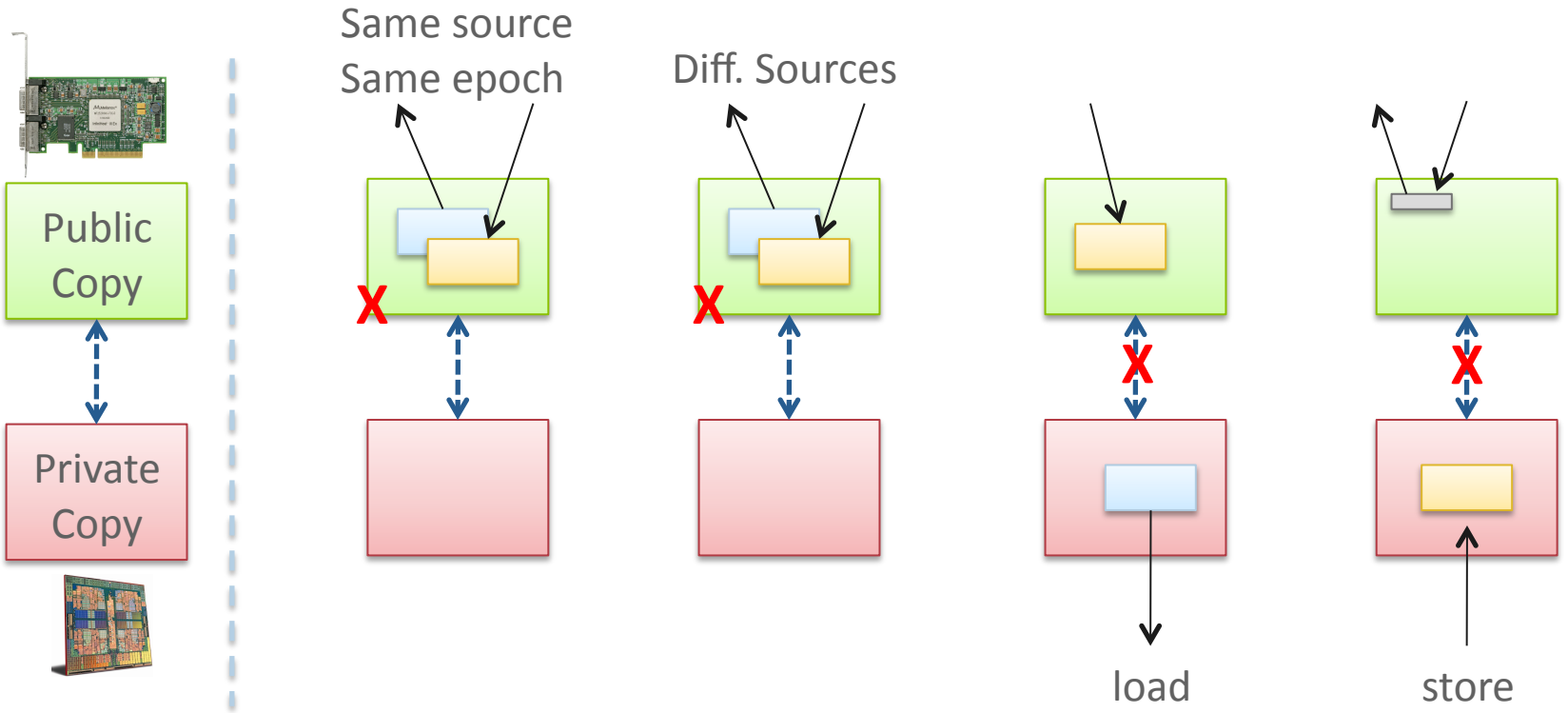
MPI Remote Memory Access Interface

- Active and Passive target Modes
  - Active: target participates
  - Passive: target does not participate
- Window: Expose memory for RMA
  - Logical public and private copies
  - Conservative data consistency model
- Accesses must occur within an epoch
  - Lock(window, rank) … Unlock(window, rank)
  - Access mode can be exclusive or shared
  - Operations are not ordered within an epoch



MPI-2 RMA "Separate" Memory Model

Concurrent, conflicting accesses are erroneous



- Conservative, but extremely portable
- Compatible with non-coherent memory systems

Translation: Global Memory Regions

- Translate between ARMCI and MPI shared data segment representations
  - ARMCI: Array of base pointers
  - MPI: Window object
- Translate between MPI and ARMCI comm. parameters
  - MPI: 〈win., win. rank, disp.〉
  - ARMCI: 〈abs. rank, address〉
- Preserve MPI RMA semantics
  - Manage access epochs
  - Avoid conflicting accesses
  - Protect shared buffers

Absolute Process Id

Metadata	0	1	…	N
0	0x6b7	0x7af		0x0c1
1	0x9d9	0x0		0x0
2	0x611	0x38b		0x659
3	0xa63	0x3f8		0x0

Allocation Metadata

```
MPI_Win    window;
int        size[nproc];
ARMCI_Group grp;
ARMCI_Mutex rmw;
…
```

GMR: Preserving MPI local access semantics

ARMCI_Put(src = 0x9e0, dst = 0x39b,
size = 8 bytes, rank = 1);

- Problem: Local buffer is also shared
  - Can't access without epoch
  - Can we lock it?
    - Same window: not allowed
    - Diff window: can deadlock
- Solution: Copy to private buffer

```
src_copy = Lock; memcpy; Unlock

xrank = GMR_Translate(comm, rank)
Lock(win, xrank)
Put(src_cpy, dst, size, xrank)
Unlock(win, xrank)
```

Absolute Process Id

Metadata	0	1	…	N
0	0x6b7	0x7af		0x0c1
1	0x9d9	0x0		0x0
2	0x611	0x38b		0x659
3	0xa63	0x3f8		0x0

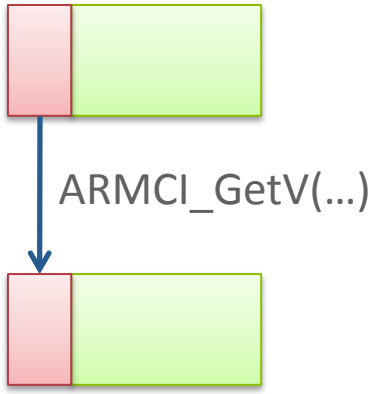
Allocation Metadata

window = win;
Size = { 1024, …};
Grp = comm;
…

ARMCI Noncontiguous Operations: I/O Vector

- Generalized noncontiguous transfer with uniform segment size:

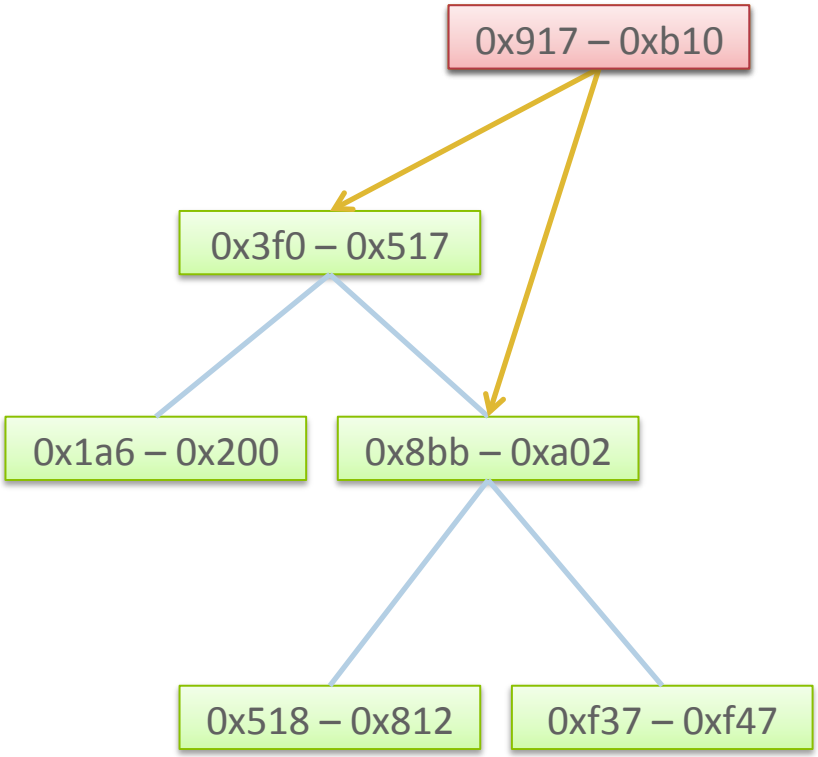
```
typedef struct {
        void **src_ptr_array;     // Source addresses
        void **dst_ptr_array;     // Dest. Addresses
        int bytes;                // Length of all seg.
        int ptr_array_len;        // Number of segments
} armci_giov_t;
```



- Three methods to support in MPI
  1. Conservative (one epoch): Lock, Put/Get/Acc, Unlock, …
  2. Batched (multiple epochs): Lock, Put/Get/Acc, …, Unlock
  3. Direct: Generate MPI indexed datatype for source and destination
     - Single operation/epoch: Lock, Put/Get/Acc, Unlock
     - Handoff processing to MPI

Avoiding concurrent, conflicting accesses

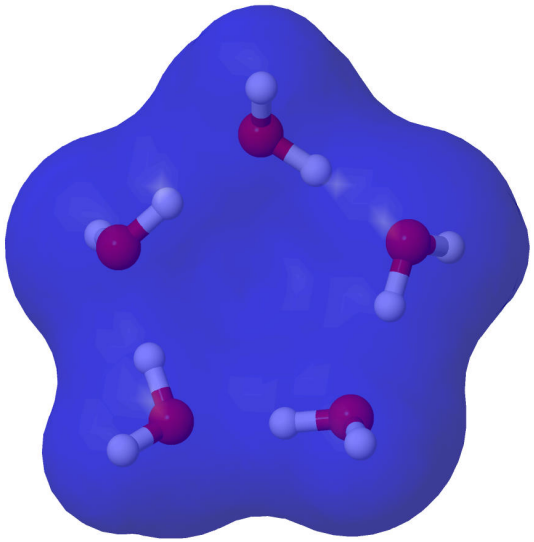
- Contiguous operations
  - Don't know what other nodes do
  - Wrap each call in an exclusive epoch
- Noncontiguous operations
  - I/O Vector segments may overlap
  - MPI Error!
- Must detect errors and fall back to conservative mode if needed
  - MPI Errors can be fatal
- Generate a conflict tree
  - Sorted, self-balancing AVL tree
  1. Search the tree for a match
  2. If (match found): Conflict!
  3. Else: Insert into the tree
- Merge search/insert steps into a single traversal



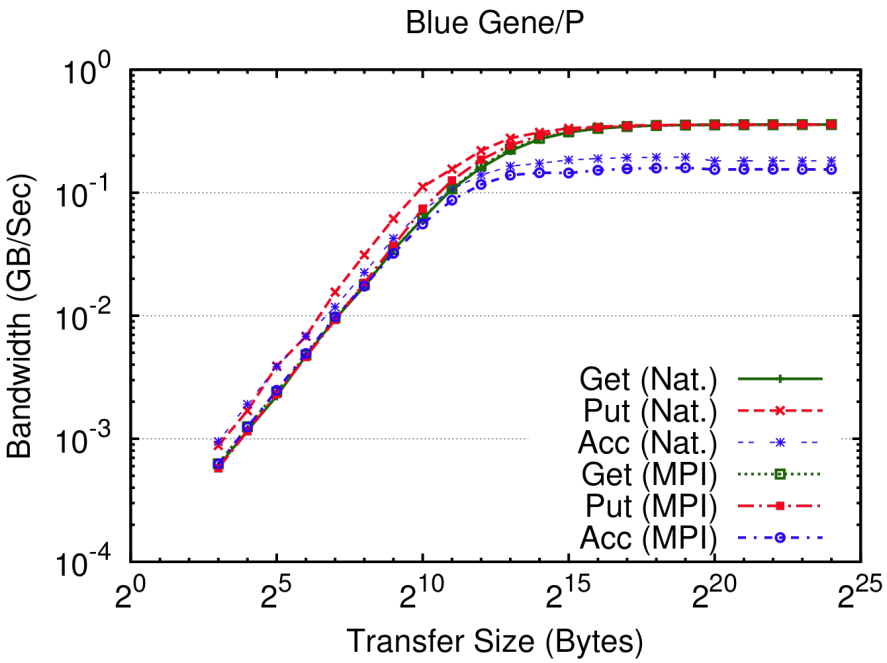
Experimental Setup

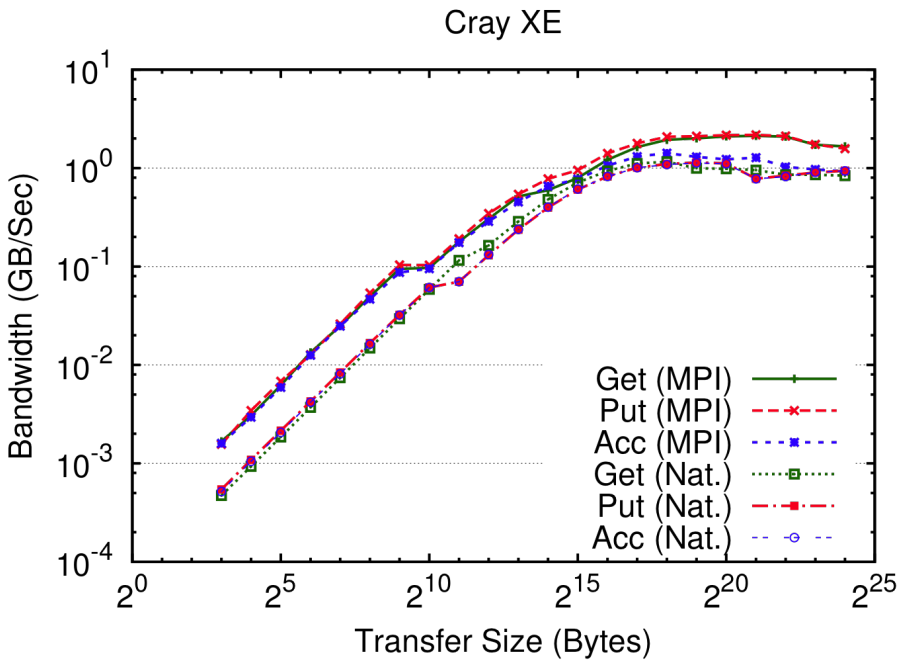
System	Nodes	Cores	Memory	Interconnect	MPI
IBM BG/P (Intrepid)	40,960	1 x 4	2 GB	3D Torus	IBM MPI
IB (Fusion)	320	2 x 4	36 GB	IB QDR	MVAPICH2 1.6
Cray XT5 (Jaguar)	18,688	2 x 6	16 GB	Seastar 2+	Cray MPI
Cray XE6 (Hopper)	6,392	2 x 12	32 GB	Gemini	Cray MPI

- Communicaton Benchmarks
  - Contiguous bandwidth
  - Noncontiguous Bandwidth
- NWChem performance evaluation
  - CCSD(T) calculation on water pentamer
- IB, XT5: Native much better than ARMCI-MPI (needs tuning/MPI-3)



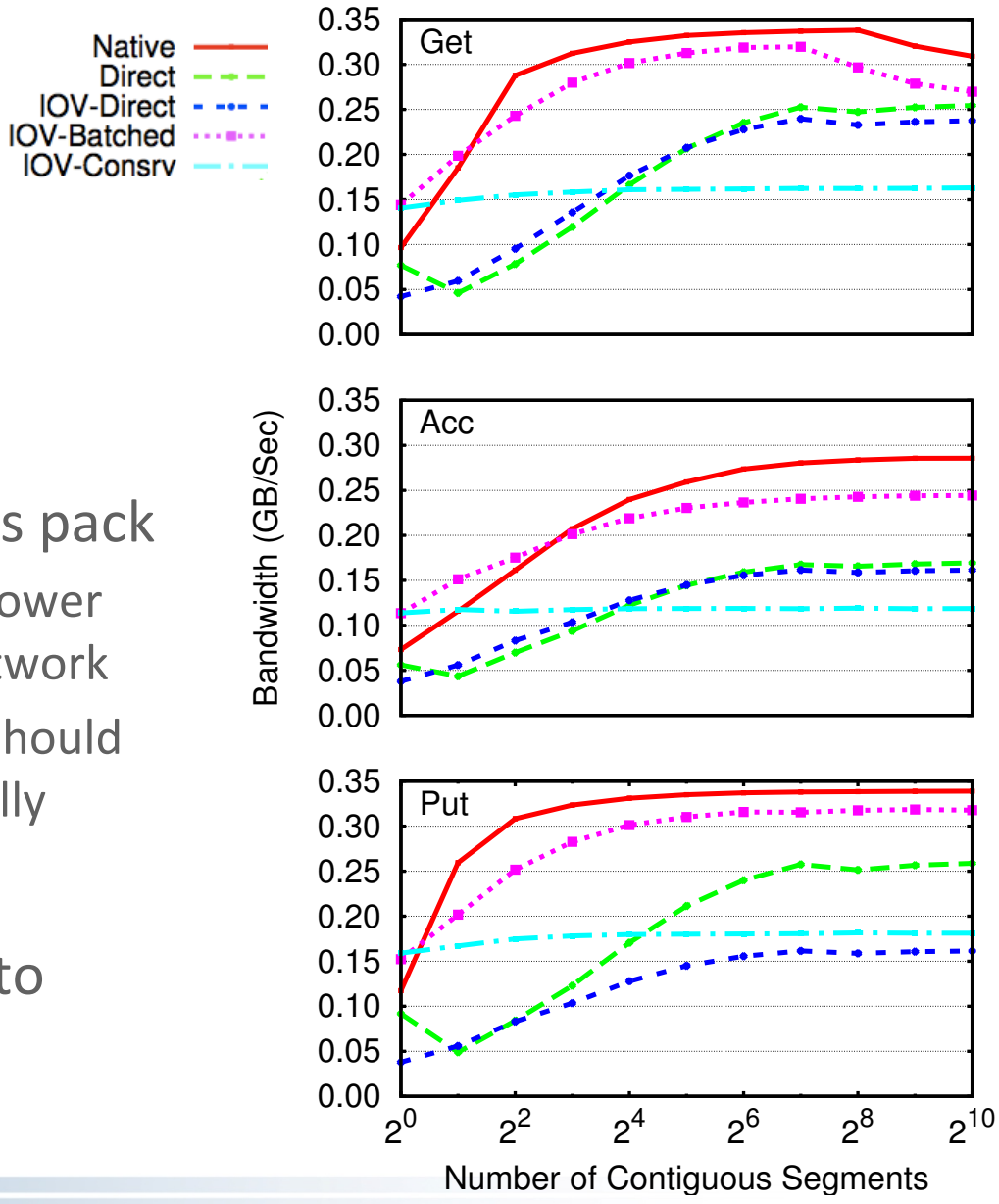
Contiguous Communication Bandwidth (BG/P & XE6)





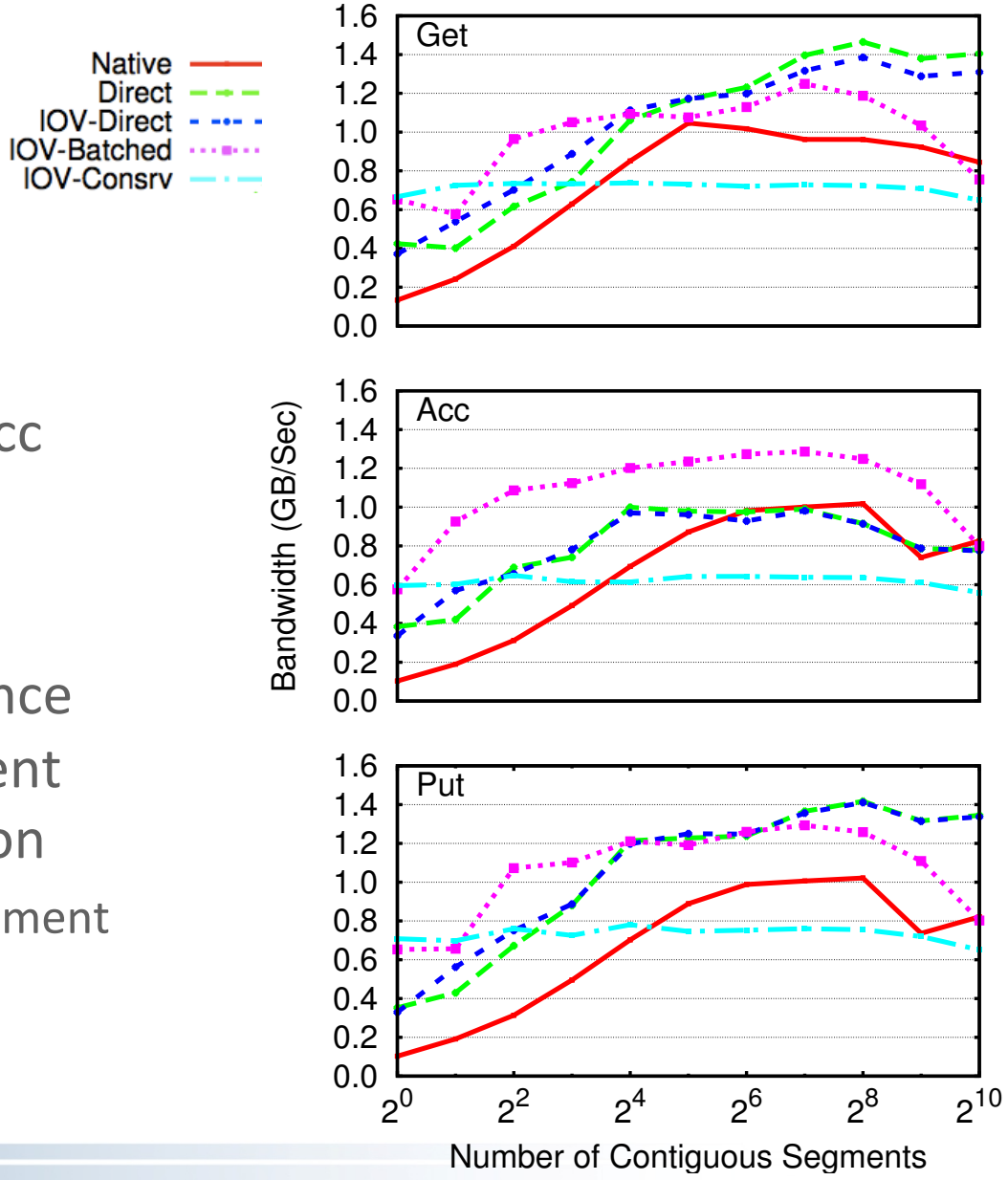
- BG/P: Native is better for small to medium size messages
  - Bandwidth regime: get/put are same and acc is ~15% less BW
- XE: ARMCI-MPI is 2x better for get/put
  - Double precision accumulate, 2x better small, same large xfers

Strided Communication Bandwidth (BG/P)



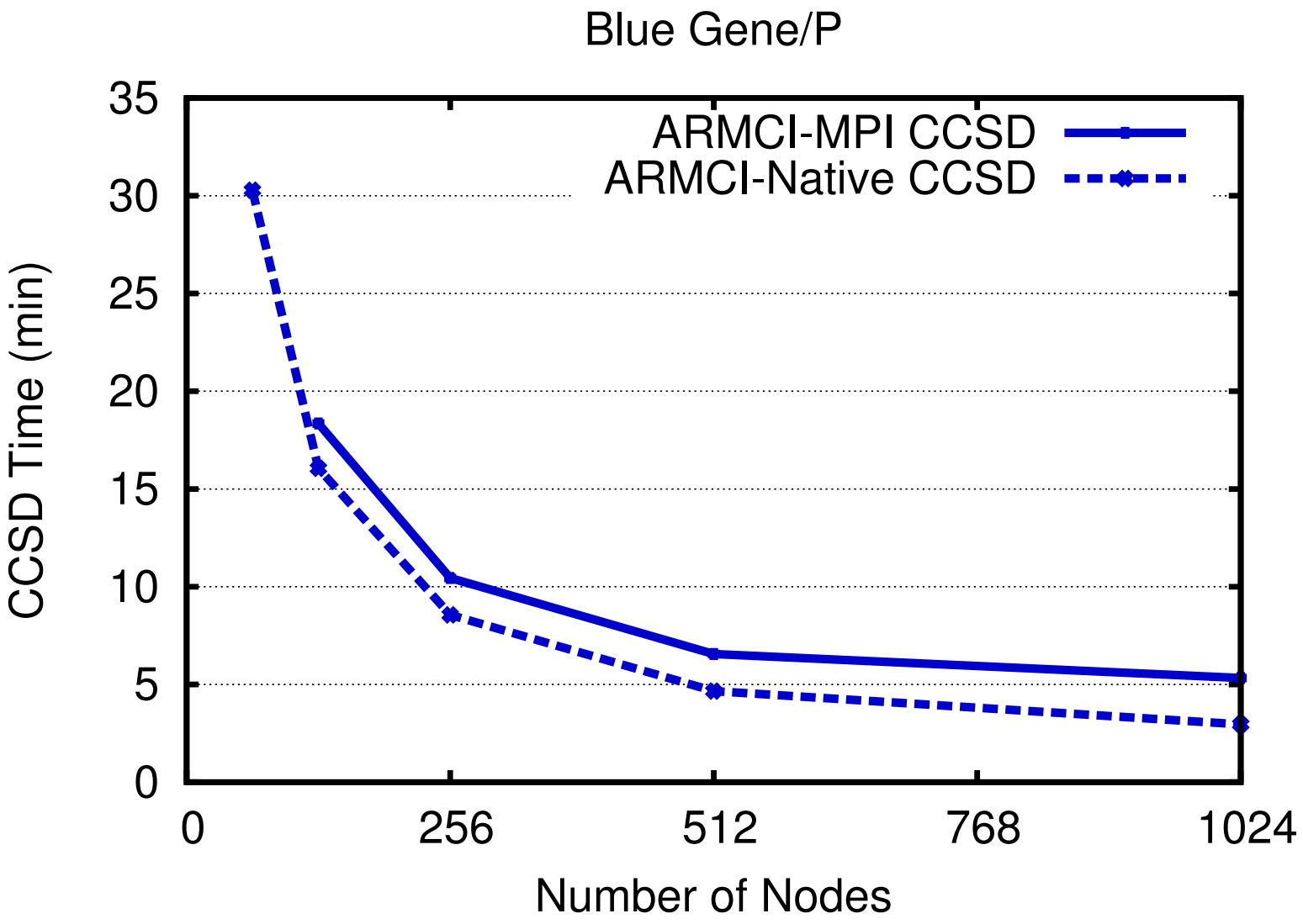
- Segment size 1 kB
- Batched is best
- Other methods always pack
  - Packing in host CPU slower than injecting into network
  - MPI implementation should select this automatically
- Performance is close to native

Strided Communication Benchmark (XE6)

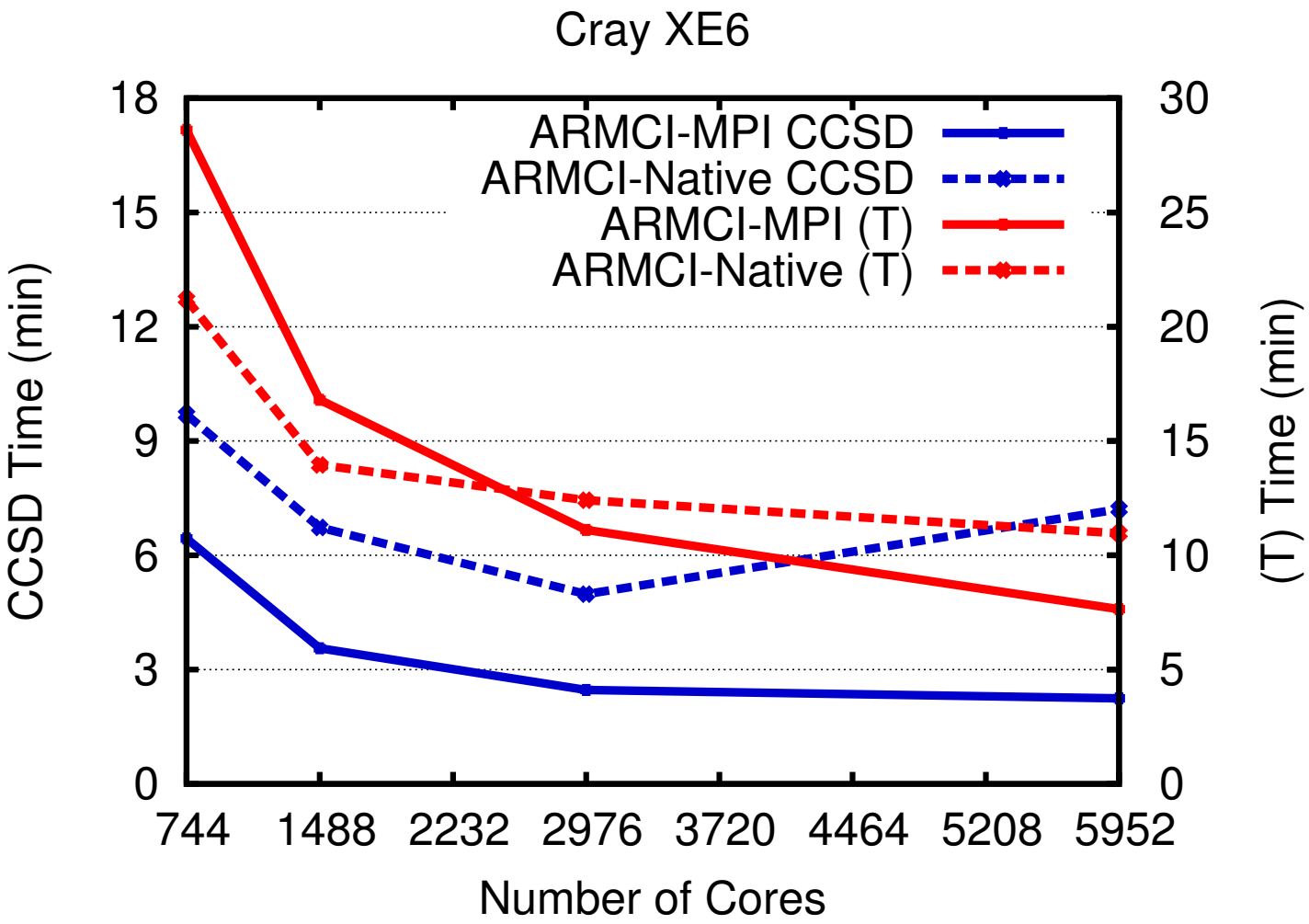


- Segment size 1 kB
- Batched is best for Acc
- Not clear for others
- Significant performance advantage over current native implementation
  - Under active development

NWChem Performance (BG/P)



NWChem Performance (XE6)

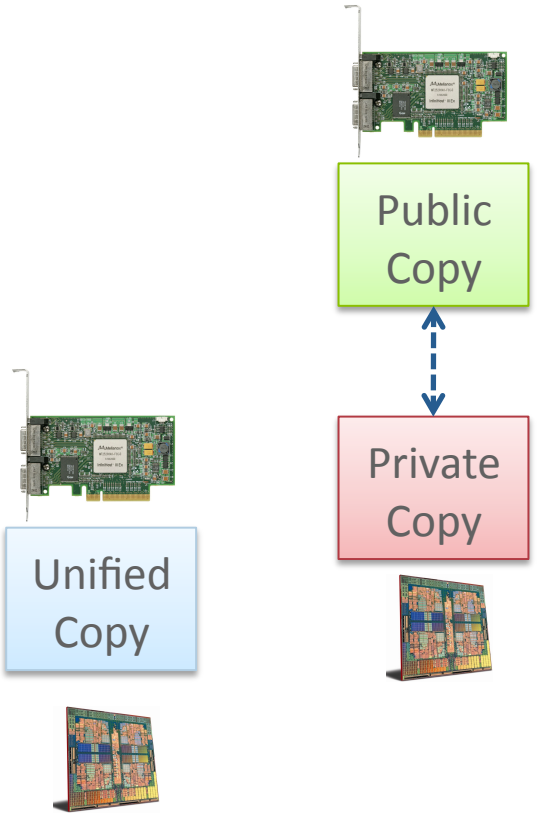


ARMCI on MPI-2 Implementation Checklist

1. One-sided
   - ✓ Translation provided by GMR
   - – Effort needed to avoid concurrent, conflicting accesses
2. Collectives
   - ✓ Implemented as MPI user-defined collectives
3. Mutexes
   - – MPI-2 RMA does not support atomic RMW operations
     - Read, modify, write is forbidden within an epoch
   - – Queueing mutex implementation (Latham, Ross, & Thakur [IJHPCA '07])
     - ✘ O(P) space
4. One-sided atomic swap, fetch-and-add (atomic w.r.t. other atomics)
   - ✘ Attach an RMW mutex to each GMR
     - Mutex_lock, get, modify, put, Mutex_unlock
     - Slow, best we can do in MPI-2
5. Non-blocking operations
   - ✘ Implemented as blocking
6. Non-collective processor groups
   - – Recursive intercomm. merging algorithm (Dinan, et al. [EuroMPI '11])
     - ✘ $O(\log^2 P)$ communication cost

New Capabilities and Features in MPI-3 RMA

- "Unified" memory model
  - Take advantage of coherent hardware
  - Relaxed synchronization will yield better performance
- Conflicting accesses
  - Localized to locations accessed
  - Relaxed to undefined
  - Load/store does not "corrupt"
- Atomic CAS, and Fetch-and-Add, and new accumulate operations
  - Mutex space overhead MPI-3: O(1)
- Request-based non-blocking ops
- Shared memory (IPC) windows



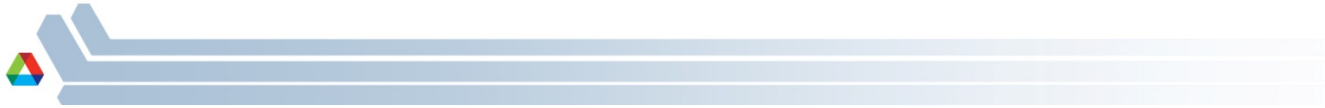
Implementation of MPI-3 RMA in MPICH

- Extensive remodeling of CH3 RMA implementation
  - New window types
  - New communication operations
  - New synchronization operations/modes
  - Passive target at multiple targets

- Created new infrastructure to enable performance research
  - Tracking of operations, window synchronization state
  - More flexible passive target sync. and completion of operations
- MPI-3 RMA is available in MPICH 3.0rc1
  - Released: Nov. 13, 2012
  - Please try it out!

ARMCI on MPI-3 Implementation (ongoing work)

1. One-sided
   - ✓ Translation provided by GMR
   - ✓ No need to avoid concurrent, conflicting accesses (undefined results)
2. Collectives
   - ✓ Implemented as MPI user-defined collectives
3. Mutexes
   - ✓ Space-scalable, using distributed MCS lock
4. One-sided atomic swap, fetch-and-add (atomic w.r.t. other atomics)
   - ✓ Implementation using MPI-3 RMW operations
5. Non-blocking operations
   - ✓ Implementation using MPI-3 request-generating operations
6. Non-collective processor groups
   - ✓ Implementation using MPI-3 MPI_Comm_create_group()

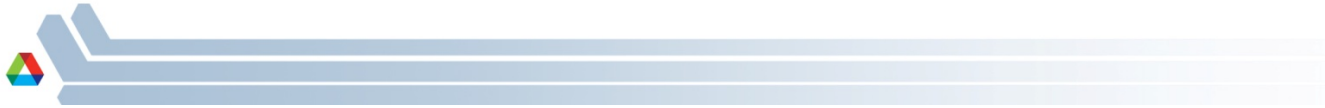


MPI-3 RMA Discussion

- Discuss opportunities for collaboration and involvement in ongoing work

- MPI-3 RMA
  - Performance optimization
    - Shared memory optimization
    - RDMA optimization
  - Development of performance and correctness debugging tools

- Global-View
  - Apply MPI-3 as a substrate for additional existing and new global-view models



Conclusions

- ARMCI-MPI:
  - Complete, portable runtime system for GA and NWChem
    - MPI-2 performance driver
    - MPI-3 feature driver
  - Mechanisms to overcome interface and semantic mismatch
  - Performance is pretty good, dependent on impl. Tuning
  - Available for download with MPICH
- MPI-3:
  - Exciting new capabilities
  - Increased flexibility
  - Avaliable in MPICH 3.0

Contact: Jim Dinan <dinan@mcs.anl.gov>