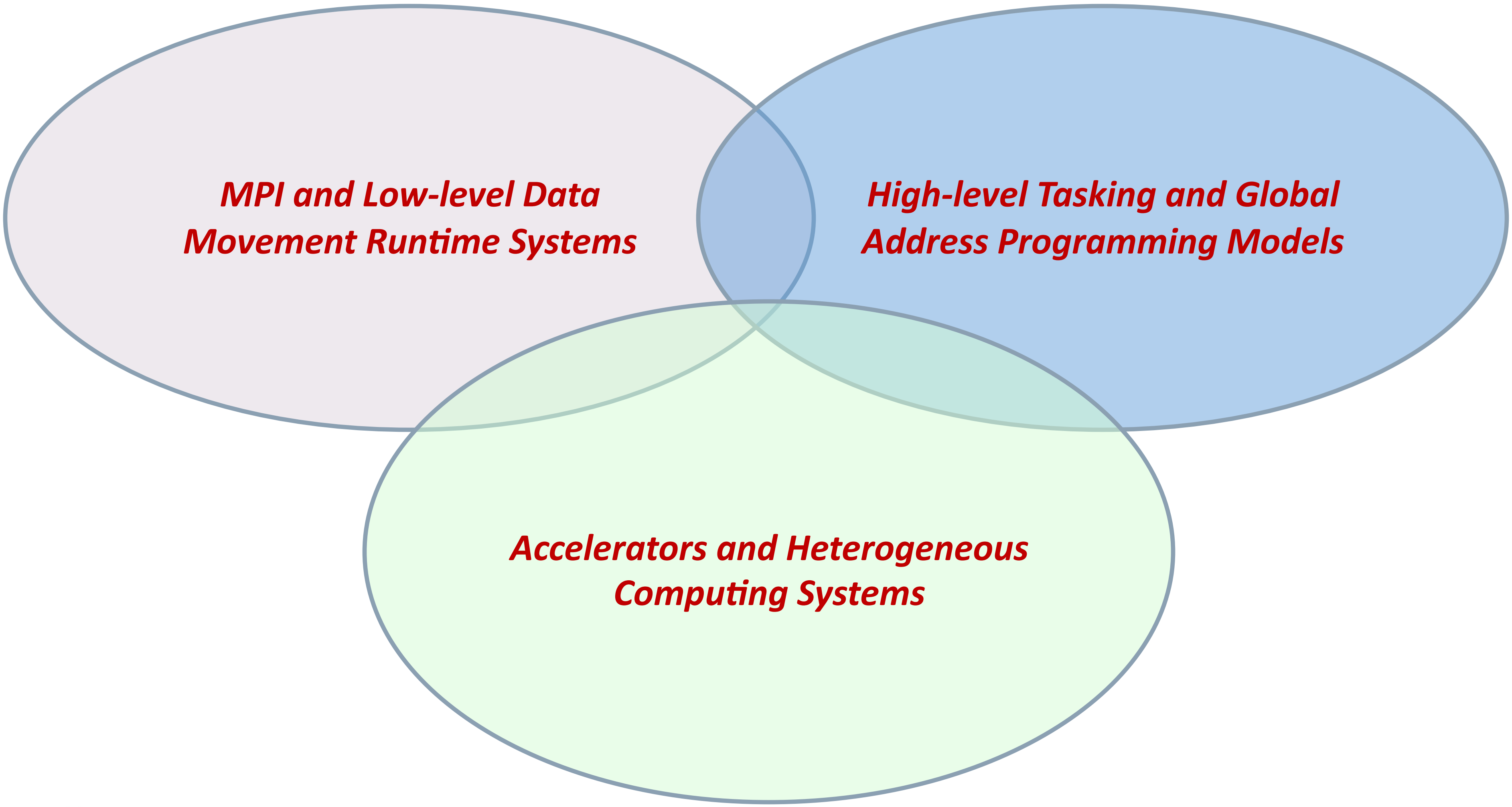
Recent Activities in Programming Models and Runtime Systems at ANL

Rajeev Thakur
Deputy Director
Mathematics and Computer Science Division
Argonne National Laboratory

Joint work with Pavan Balaji, Darius Buntinas, Jim Dinan, Dave Goodell, Bill Gropp, Rusty Lusk, Marc Snir

Programming Models and Runtime Systems Efforts at Argonne



MPI at Exascale

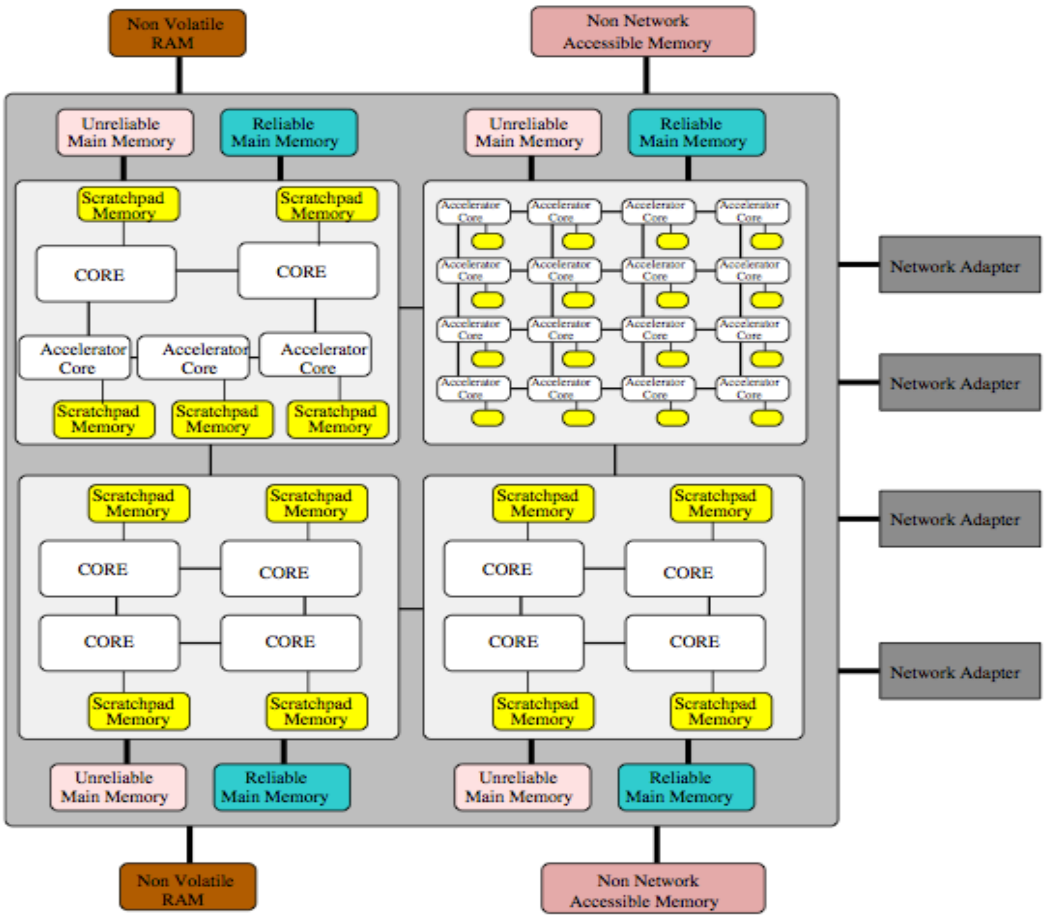
- We believe that MPI has a role to play even at exascale, particularly in the context of an MPI+X model
- The investment in application codes and libraries written using MPI is on the order of hundreds of millions of dollars
- Until a viable alternative to MPI for exascale is available, and applications have been ported to the new model, MPI must evolve to run as efficiently as possible on future systems
- Argonne has long-term strength in all things related to MPI

MPI and Low-level Runtime Systems

- MPICH implementation of MPI
  - Scalability of MPI to very large systems
    - Scalability improvements to the MPI implementation
    - Scalability to large numbers of threads
  - MPI-3 algorithms and interfaces
    - Remote memory access
    - Fault Tolerance
  - Research on features for future MPI standards
    - Interaction with accelerators
    - Interaction with other programming models
- Data movement models on complex processor and memory architectures
  - Data movement for heterogeneous memory (accelerator memory, NVRAM)

Pavan Balaji wins DOE Early Career Award

- Pavan Balaji was recently awarded the highly competitive and prestigious DOE early career award ($500K/yr for 5 years)
- "Exploring Efficient Data Movement Strategies for Exascale Systems with Deep Memory Hierarchies"



MPICH-based Implementations of MPI

- IBM MPI for the Blue Gene/Q
  - IBM has successfully scaled the LAMMPS application to over 3 million MPI ranks and submitted a special Gordon Bell prize entry
    - ***So it's not true that MPI won't scale!***
  - Will be used on Livermore (Sequoia), Argonne (Mira), and other major BG/Q installations
- Unified IBM implementation for BG and Power systems
- Cray MPI for XE/XK-6
  - On Blue Waters, Oak Ridge (Jaguar, Titan), NERSC (Hopper), HLRS Stuttgart, and other major Cray installations
- Intel MPI for clusters
- Microsoft MPI
- Myricom MPI
- MVAPICH2 from Ohio State

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

Accelerators and Heterogeneous Computing Systems

- Accelerator-augmented data movement models
  - Integrated data movement models that can move data from any memory region of a given process to any other memory region of another process
    - E.g., move data from accelerator memory of one process to that of another
  - Internal runtime optimizations for efficient data movement
  - External programming model improvements for improved productivity
- Virtualized accelerators
  - Allow applications to (semi-)transparently utilize light-weight virtual accelerators instead of physical accelerators
  - Backend improvements for performance, power, fault tolerance, etc.

Recent Work

1. MPI-ACC (ANL, NCSU, VT)
   - Integrate awareness of accelerator memory in MPICH2
   - Productivity and performance benefits
   - *To be presented at HPCC 2012 Conference, Liverpool, UK, June 2012*

2. Virtual OpenCL (ANL, VT, SIAT CAS)
   - OpenCL implementation allows program to use remote accelerators
   - One-to-many model, better resource usage, load balancing, FT, …
   - *Published at CCGrid 2012, Ottawa, Canada, May 2012*

3. Scioto-ACC (ANL, OSU, PNNL)
   - Task parallel programming model, scalable runtime system
   - Coscheduling CPU and GPU, automatic data movement

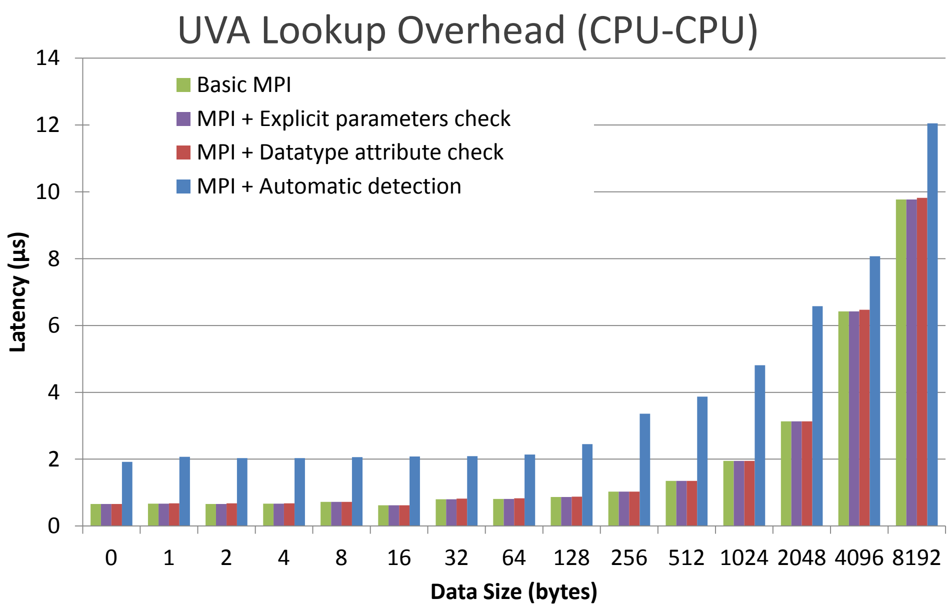
Current MPI+GPU Programming

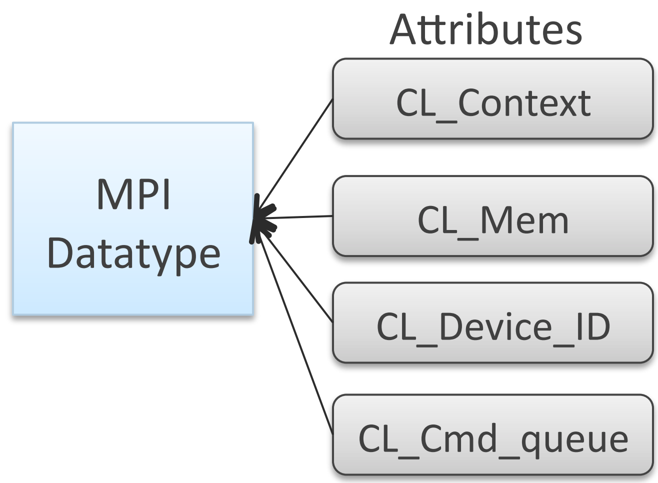
```
double *dev_buf, *host_buf;
cudaMalloc(&dev_buf, size);
cudaMallocHost(&host_buf, size);

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



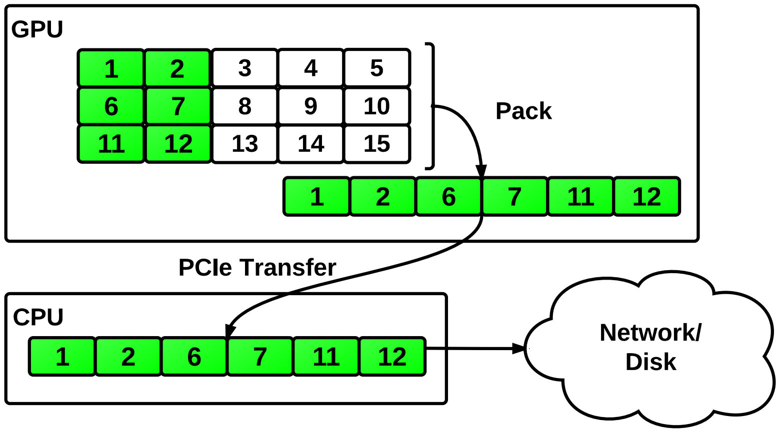


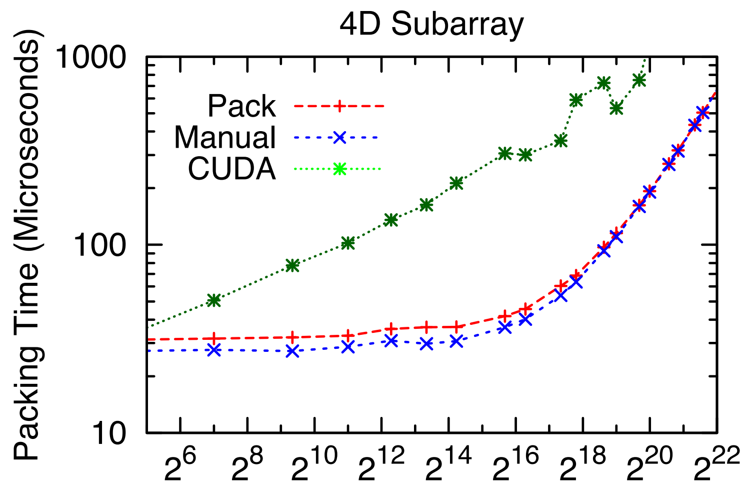
- Unified Virtual Address (UVA) space
  - Allow device pointer in MPI routines directly
  - Currently supported only by CUDA and newer NVIDIA GPUs
  - Query cost is high and added to *every* operation (CPU-CPU)
- Explicit Interface – e.g. MPI_CUDA_Send(…)
- MPI Datatypes – Compatible with MPI *and* many 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:
  - 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
    - Sender and receiver drive independent DMA transfers
  - Direct DMA protocol
    - GPU-GPU DMA transfer (CUDA IPC)
  - Both protocols needed, PCIe limitations serialize DMA across I/O hubs

Integrated Support for User-Defined Datatypes

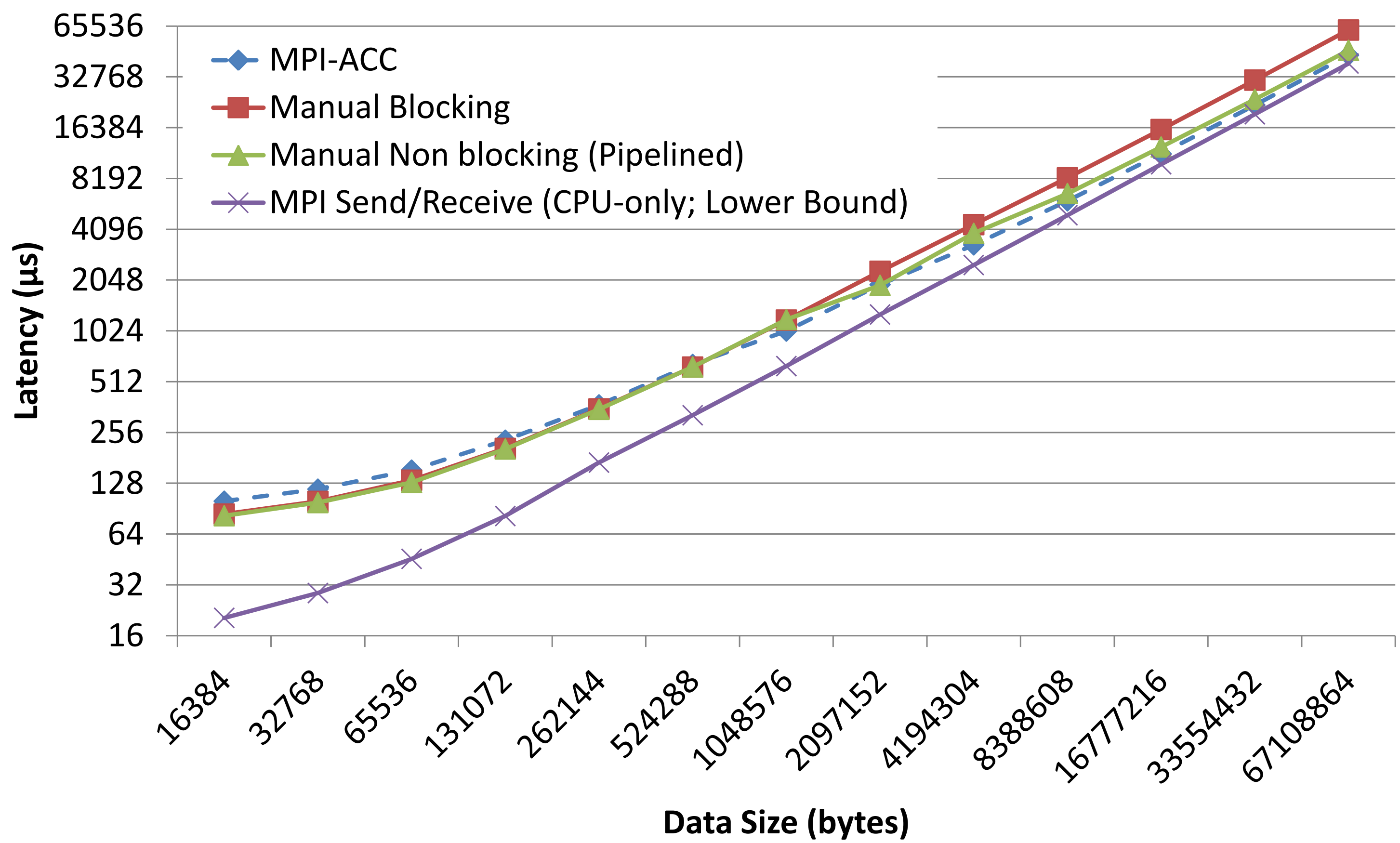





```
MPI_Send(buffer, datatype, count, to, … )
MPI_Recv(buffer, datatype, count, from, … )
```

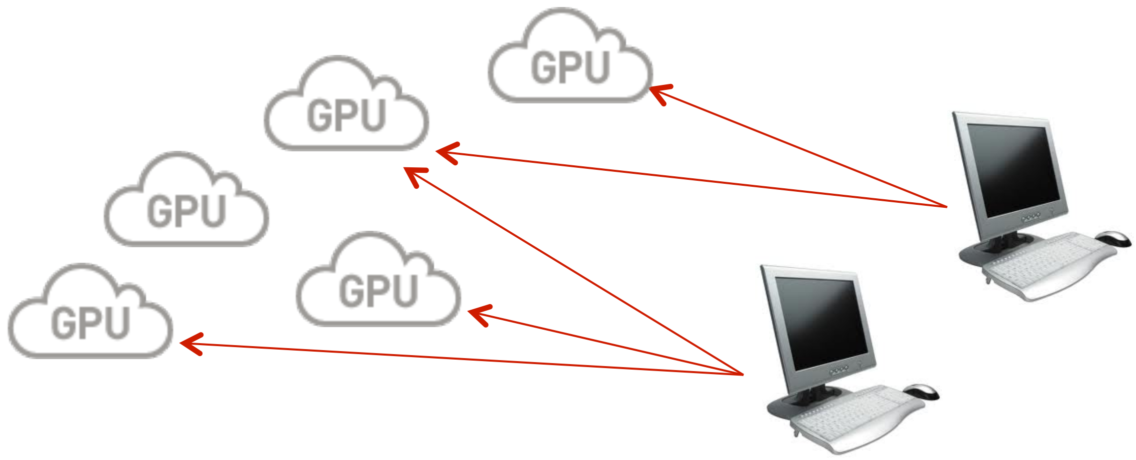
- What if the datatype is noncontiguous?
- CUDA doesn't support arbitrary noncontiguous transfers
- Pack data on the GPU
  - Flatten datatype tree representation
  - Packing kernel that can saturate memory bus/banks

Inter-node GPU-GPU Latency



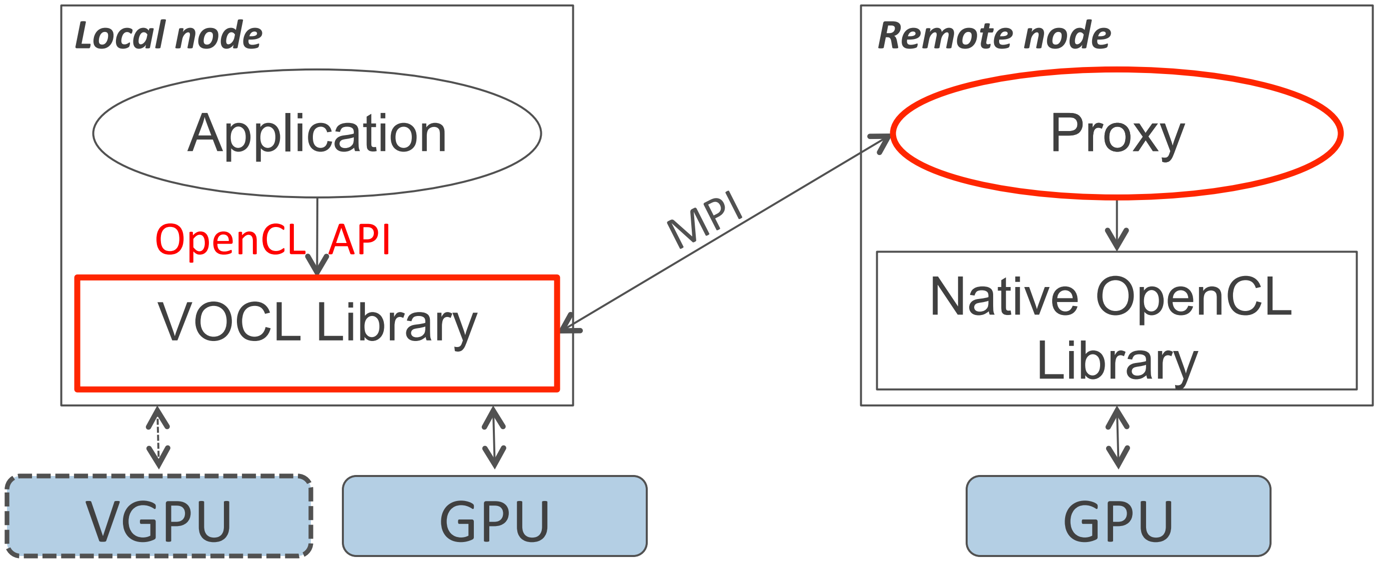
- Pipelining (PCIe + IB) pays off for large messages – 2/3 latency

GPUs as a Service: Virtual OpenCL



- Clusters, cloud systems, provide GPUs on subset of nodes
- OpenCL provides a powerful abstraction
  - Kernels compiled on-the-fly – i.e. at the device
  - Enable transparent virtualization, even across different devices
- Support GPUs as a service
  - One-to-Many: One process can drive many GPUs
  - Resource Utilization: Share GPU across applications, use hybrid nodes
  - System Management, Fault Tolerance: Transparent migration

Virtual OpenCL (VOCL) Framework Components

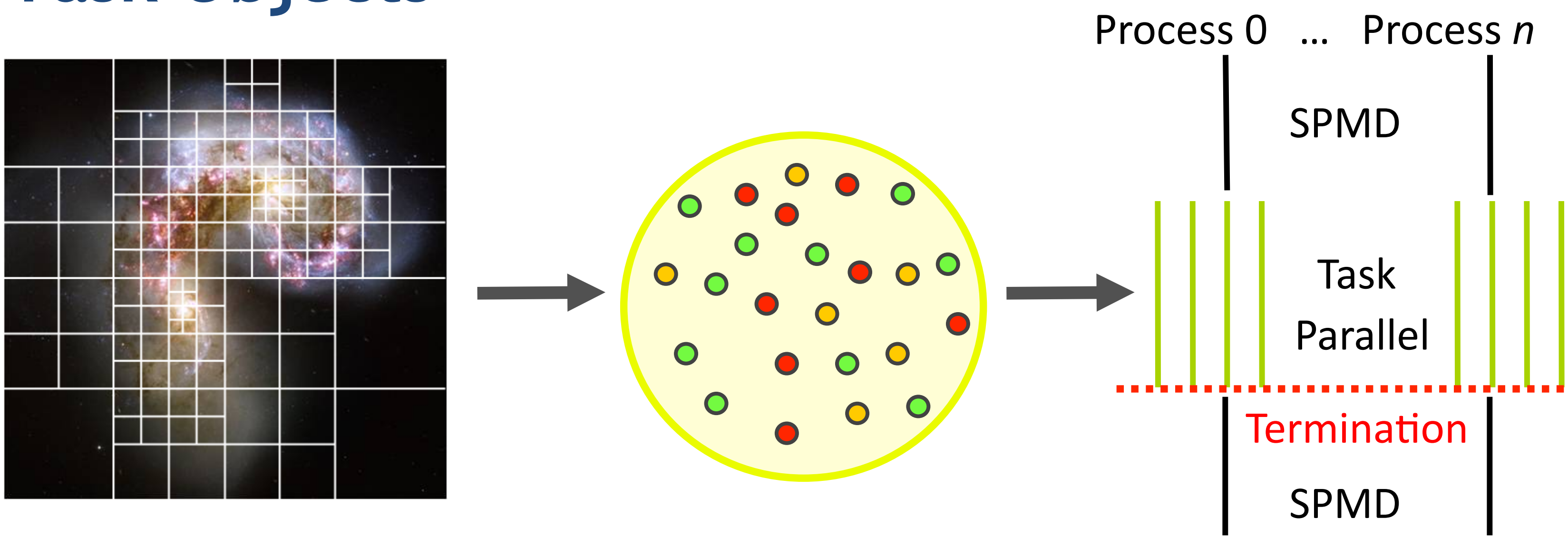


- VOCL library (local) and proxy process (system service)
- API and ABI compatible with OpenCL – transparent to app.
- Utilize both local and remote GPUs
  - Local GPUs: Call native OpenCL functions
  - Remote GPUs: Runtime uses MPI to forward function calls to proxy

High-level Tasking and Global Address Programming Models

- Led by Jim Dinan
- High-level tasking execution model for performing computations in dynamic environments
  - Programming model interface and implementation
  - Multiple programming model implementations for this tasking model to work with MPI, PGAS models, etc.
- Interoperable global address space and tasking models
  - Unified runtime infrastructure
- Benchmarking
  - Unbalanced Tree Search family of benchmarks

Scioto-ACC: Accelerated, Scalable Collections of Task Objects



- Express computation as collection of tasks
  - Tasks operate on data in Global Address Space (GA, MPI-RMA, …)
  - Executed in collective task parallel phases
- Scioto runtime system manages task execution
  - Load balancing, locality opt., fault resilience
  - Mapping to Accelerator/CPU, data movement

Led by Jim Dinan

Other Recent Papers

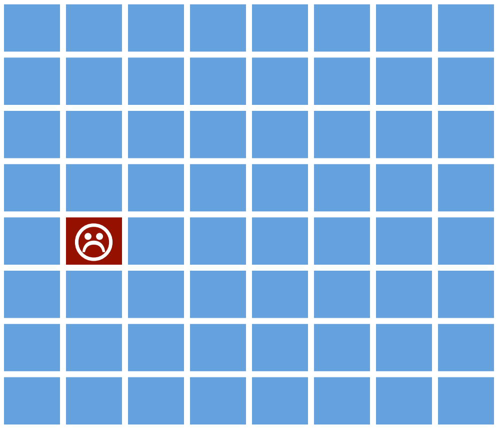
- "Scalable Distributed Consensus to Support MPI Fault Tolerance"
  - Darius Buntinas, IPDPS 2012, Shanghai, China, May 2012
  - Scalable algorithm for a set of MPI processes to collectively determine which subset of processes among them have failed

- "Supporting the Global Arrays PGAS Model Using MPI One-Sided Communication"
  - James Dinan, Pavan Balaji, Jeff Hammond, Sriram Krishnamoorthy, Vinod Tipparaju, IPDPS 2012, Shanghai, China, May 2012
  - Implements ARMCI interface over MPI one-sided

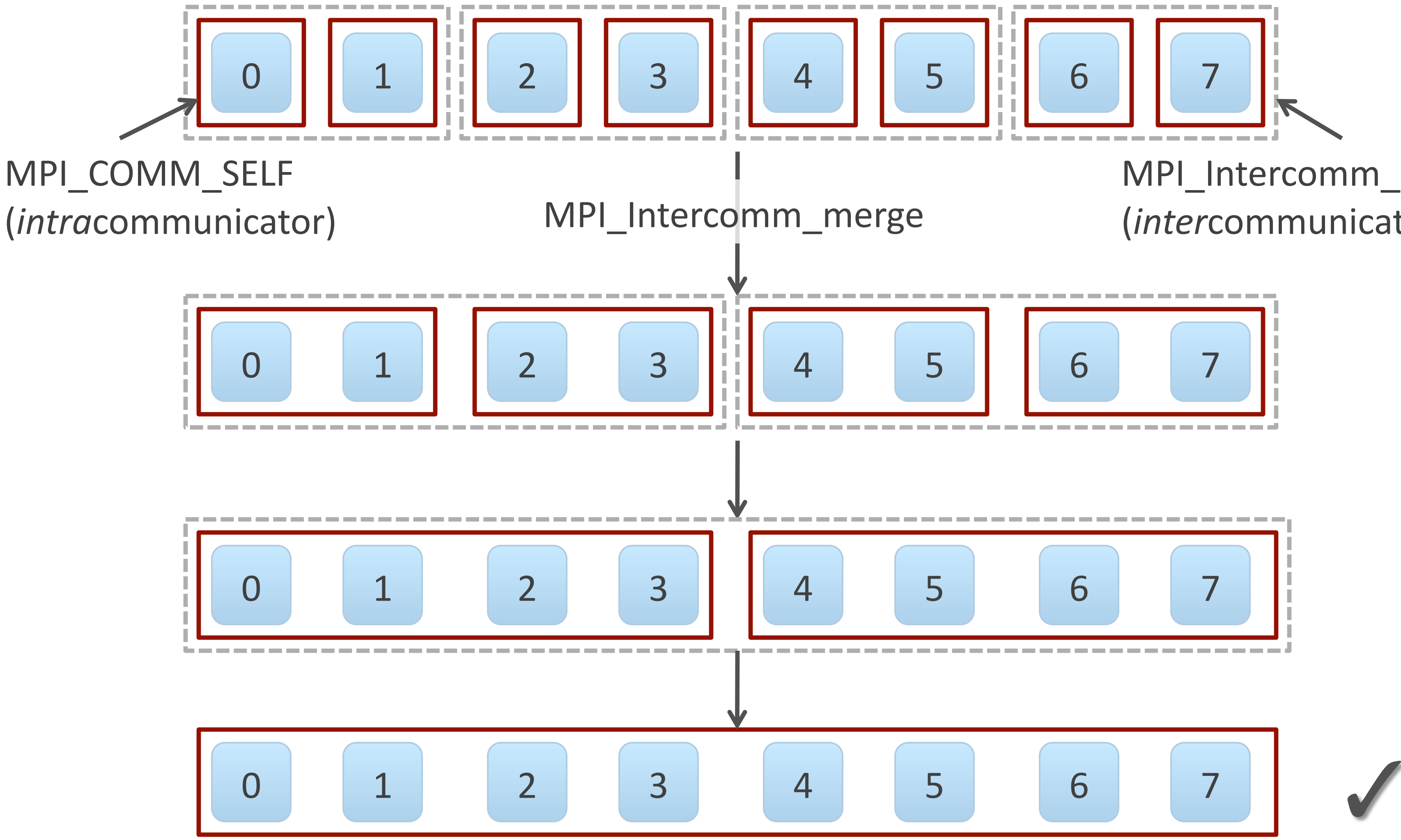
- "Efficient Multithreaded Context ID Allocation in MPI"
  - James Dinan, David Goodell, William Gropp, Rajeev Thakur, Pavan Balaji, submitted to EuroMPI 2012
  - Clever algorithm for multithreaded context id generation for MPI_Comm_create_group (new function in MPI-3)

Non-Collective Communicator Creation

- Create a communicator collectively only on new members
- Global Arrays process groups
  - Past: collectives using MPI Send/Recv
- Overhead reduction
  - Multi-level parallelism
  - Small communicators when parent is large
- Recovery from failures
  - Not all ranks in parent can participate
- Load balancing



Non-Collective Communicator Creation Algorithm (using MPI 2.2 functionality)



Non-Collective Algorithm in Detail

INPUT: $group, comm, tag$
OUTPUT: $comm'$
REQUIRE: $group$ is ordered by desired rank in $comm'$ and is identical on all callers
LET: $grp\_pids[0..|group|-1] = \mathbb{N}$ and $pids[\,]$ be arrays of length $|group|$

MPI_Comm_rank($comm$, &$rank$)
MPI_Group_rank($group$, &grp_rank), MPI_Group_size($group$, &grp_size)
MPI_Comm_dup(MPI_COMM_SELF, &$comm'$)

MPI_Comm_group($comm$, &$parent\_grp$)
MPI_Group_translate_ranks($group$, grp_size, grp_pids, $parent\_grp$, $pids$)
MPI_Group_free(&$parent\_grp$)

for ($merge\_sz \leftarrow 1$; $merge\_sz < grp\_size$; $merge\_sz \leftarrow merge\_sz \cdot 2$) **do**
  $gid \leftarrow grp\_rank / merge\_sz$, $comm\_old \leftarrow comm'$
  if $gid \mod 2 = 0$ **then**
    if $((gid+1) \cdot merge\_sz < grp\_size$ **then**
      MPI_Intercomm_create($comm'$, 0, $comm$, $pids[(gid+1) \cdot merge\_sz]$, tag, &ic)
      MPI_Intercomm_merge(ic, 0 /* LOW */, &$comm'$)
    end if
  else
    MPI_Intercomm_create($comm'$, 0, $comm$, $pids[(gid-1) \cdot merge\_sz]$, tag, &ic)
    MPI_Intercomm_merge(ic, 1 /* HIGH */, &$comm'$)
  end if
  if $comm' \neq comm\_old$ **then**
    MPI_Comm_free(&ic)
    MPI_Comm_free(&$comm\_old$)
  end if
end for

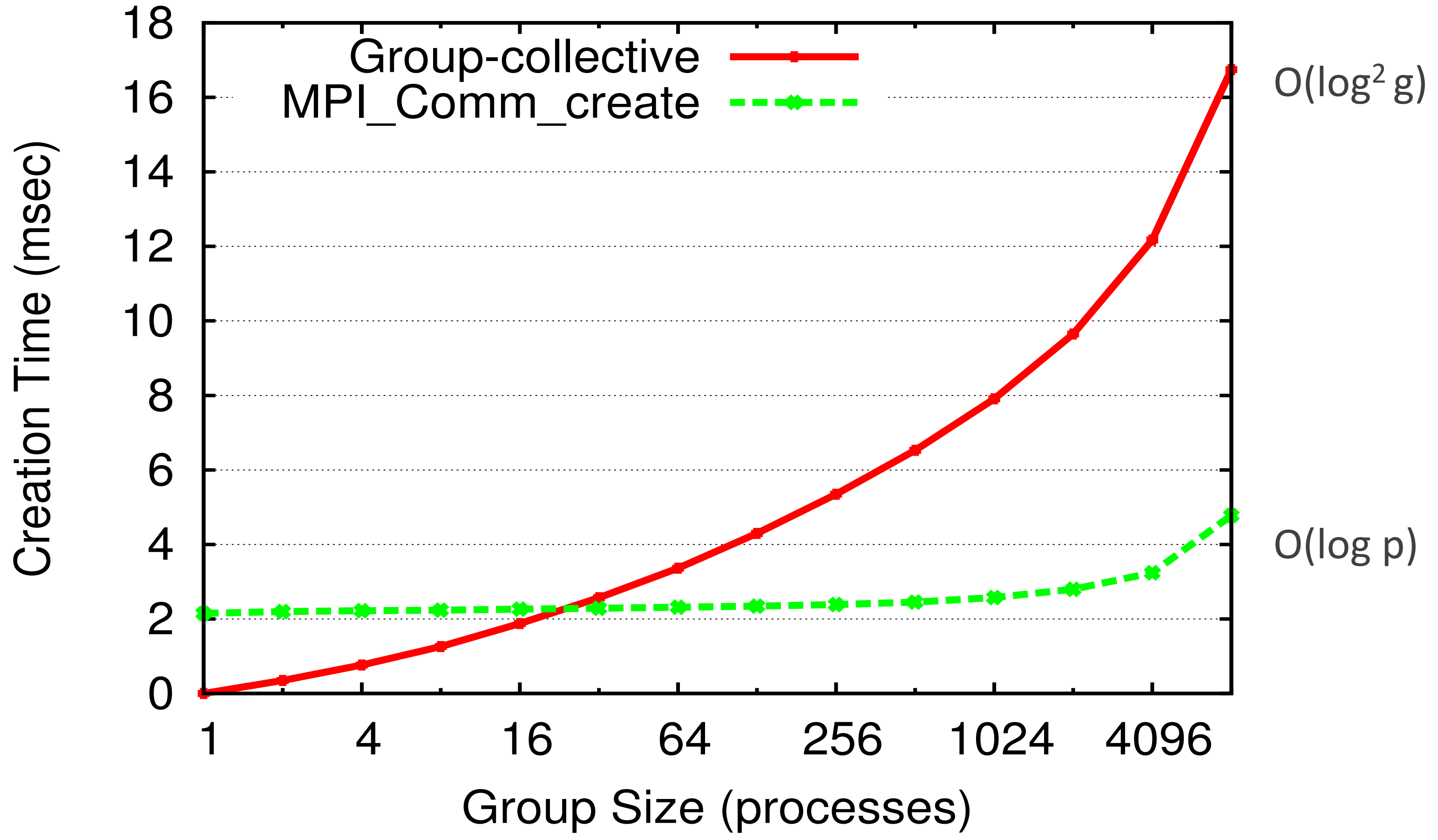
Translate group ranks to ordered list of ranks on parent communicator

Calculate my group ID

Left group

Right group

Evaluation: Microbenchmark



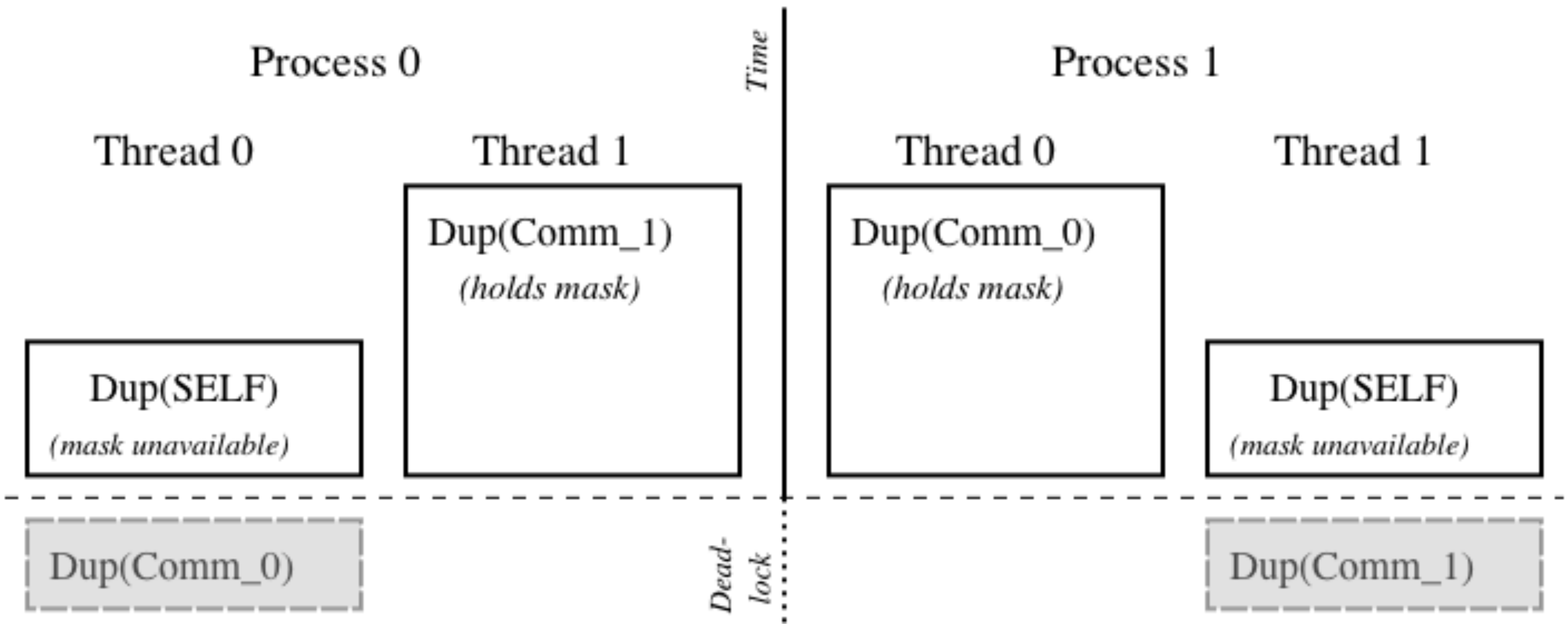
New MPI-3 Function MPI_Comm_create_group

MPI_Comm_create_group(MPI_Comm in, MPI_Group grp, int tag,
    MPI_Comm *out)

  - Collective only over process in "grp"
  - Tag allows for safe communication and distinction of calls

- Eliminate O(log g) factor by direct implementation

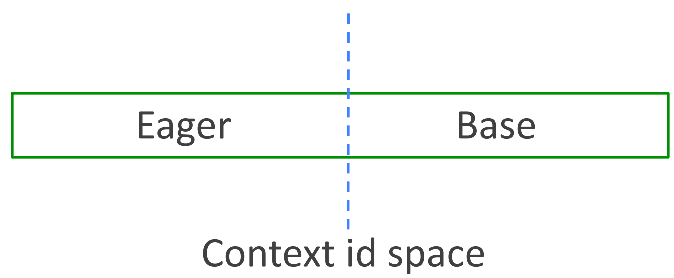
Deadlock scenario in existing multithreaded context id generation algorithm in MPICH2



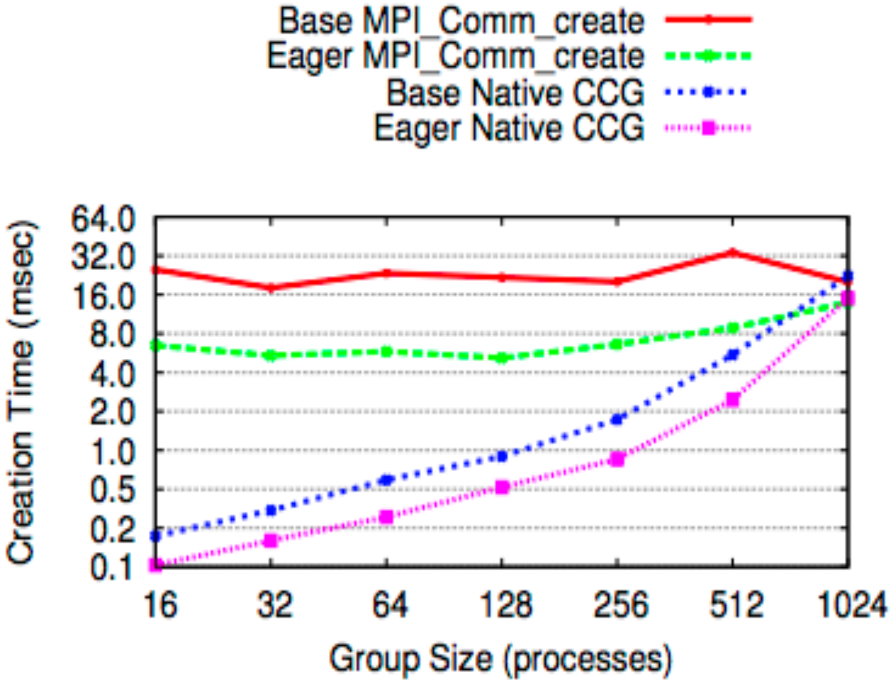
Avoiding the deadlock

- Simple solution: Add a barrier on entry to the algorithm
- However, it is additional synchronization overhead

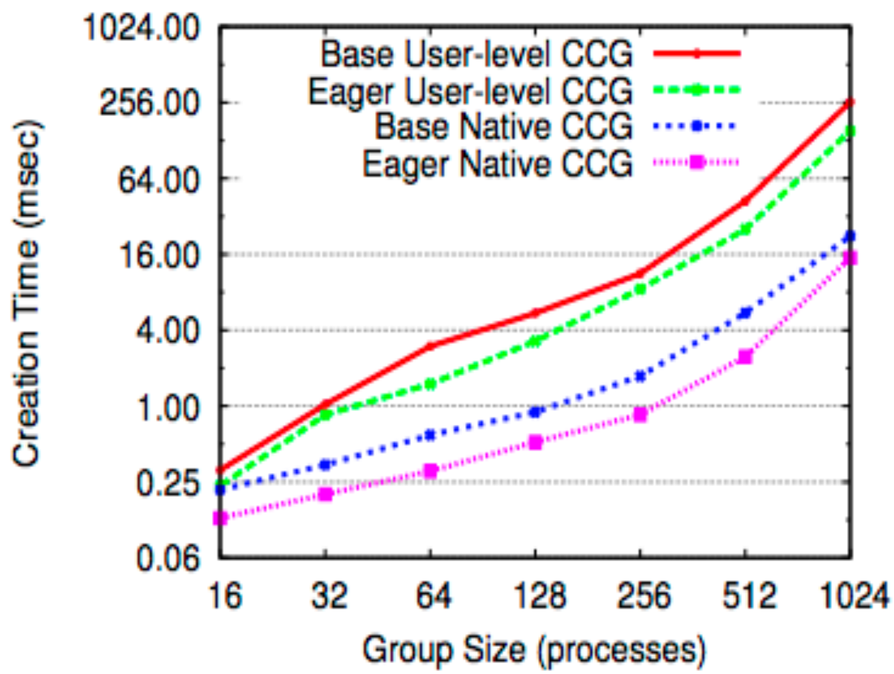
- Better solution
  - Use the synchronization to attempt to acquire a context id
  - Partition the context id space into two segments: eager and base
  - Instead of the barrier, do an Allreduce on the eager segment
  - If eager allocation fails, the allreduce acts like a barrier, and the context id is acquired in the second call to allreduce
  - If eager allocation succeeds (which in most cases it will), the overhead of an additional barrier is avoided



Performance



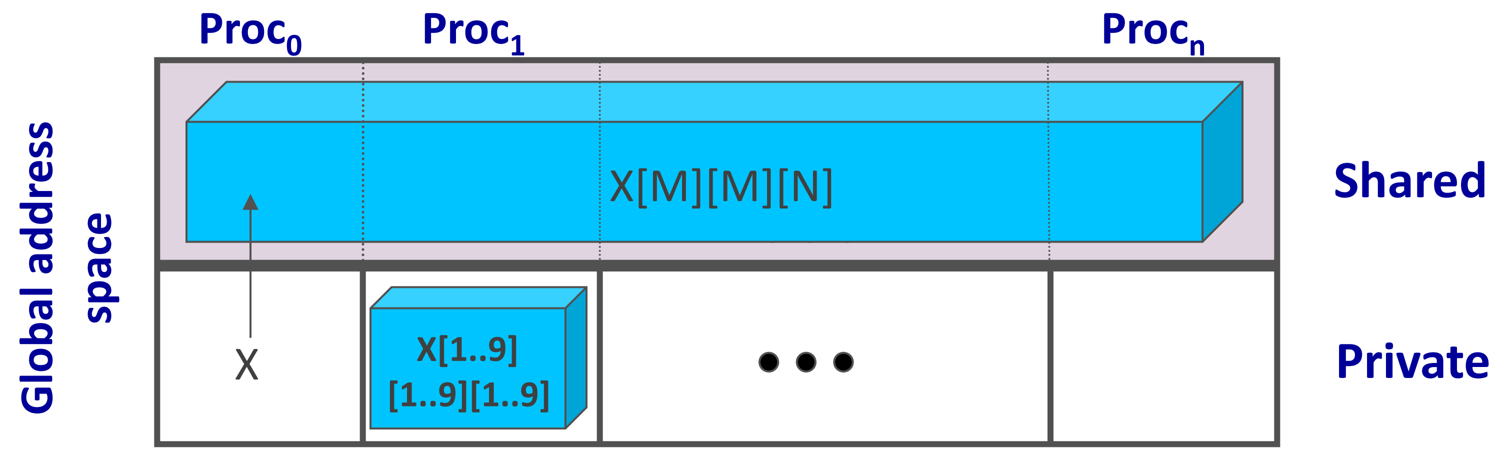
(a) Comparison with MPI_Comm_create



(b) Comparison with User-level CCG

Global Arrays over MPI One sided

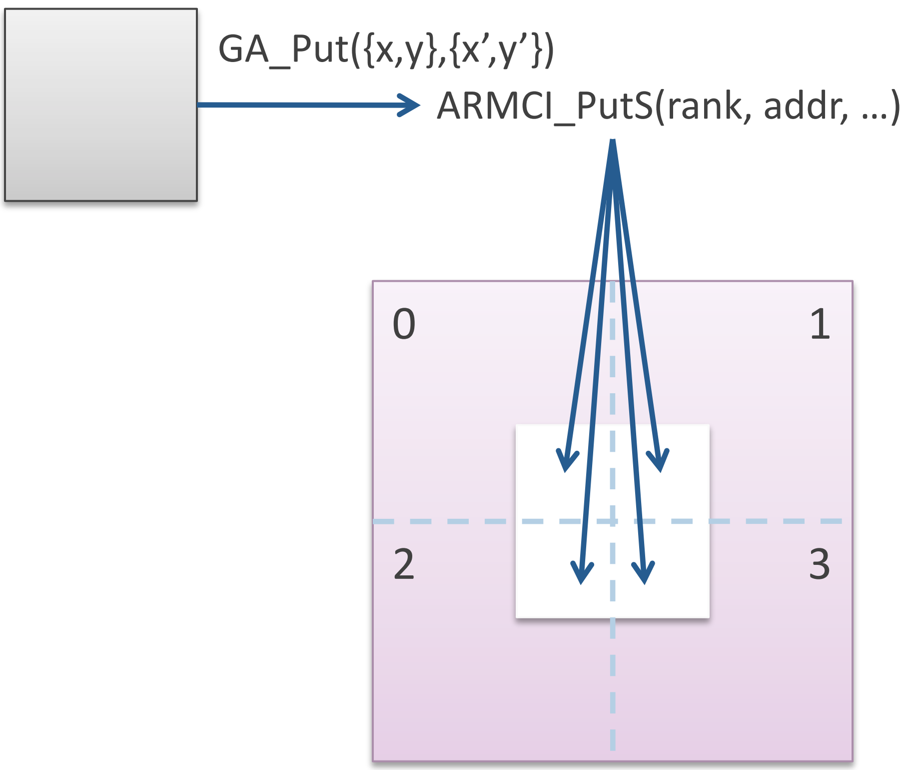
Global Arrays, a Global-View Data Model



- Distributed, shared multidimensional arrays
  - Aggregate memory of multiple nodes into global data space
  - Programmer controls data distribution, can exploit locality
- One-sided data access: Get/Put({i, j, k}…{i', j', k'})
- NWChem data management: Large coeff. tables (100GB+)

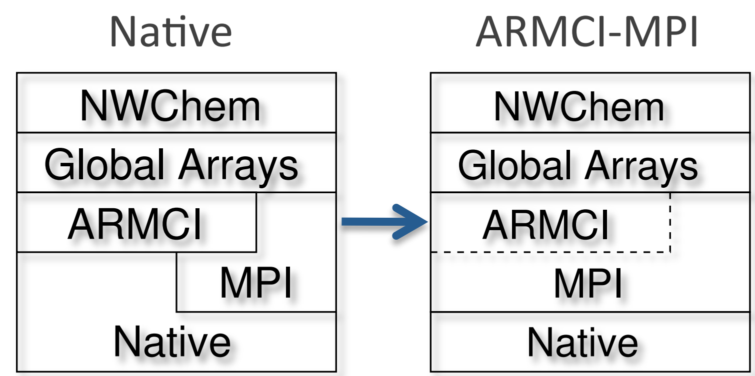
ARMCI: The Aggregate Remote Memory Copy Interface

- GA runtime system
  - Manages shared memory
  - Provides portability
  - Native implementation
- One-sided communication
  - *Get, put, accumulate, …*
  - Load/store on local data
  - Noncontiguous operations
- Mutexes, atomics, collectives, processor groups, …
- Location consistent data access
  - I see my operations in issue order



Implementing ARMCI

- 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



ARMCI/MPI-RMA Mismatch

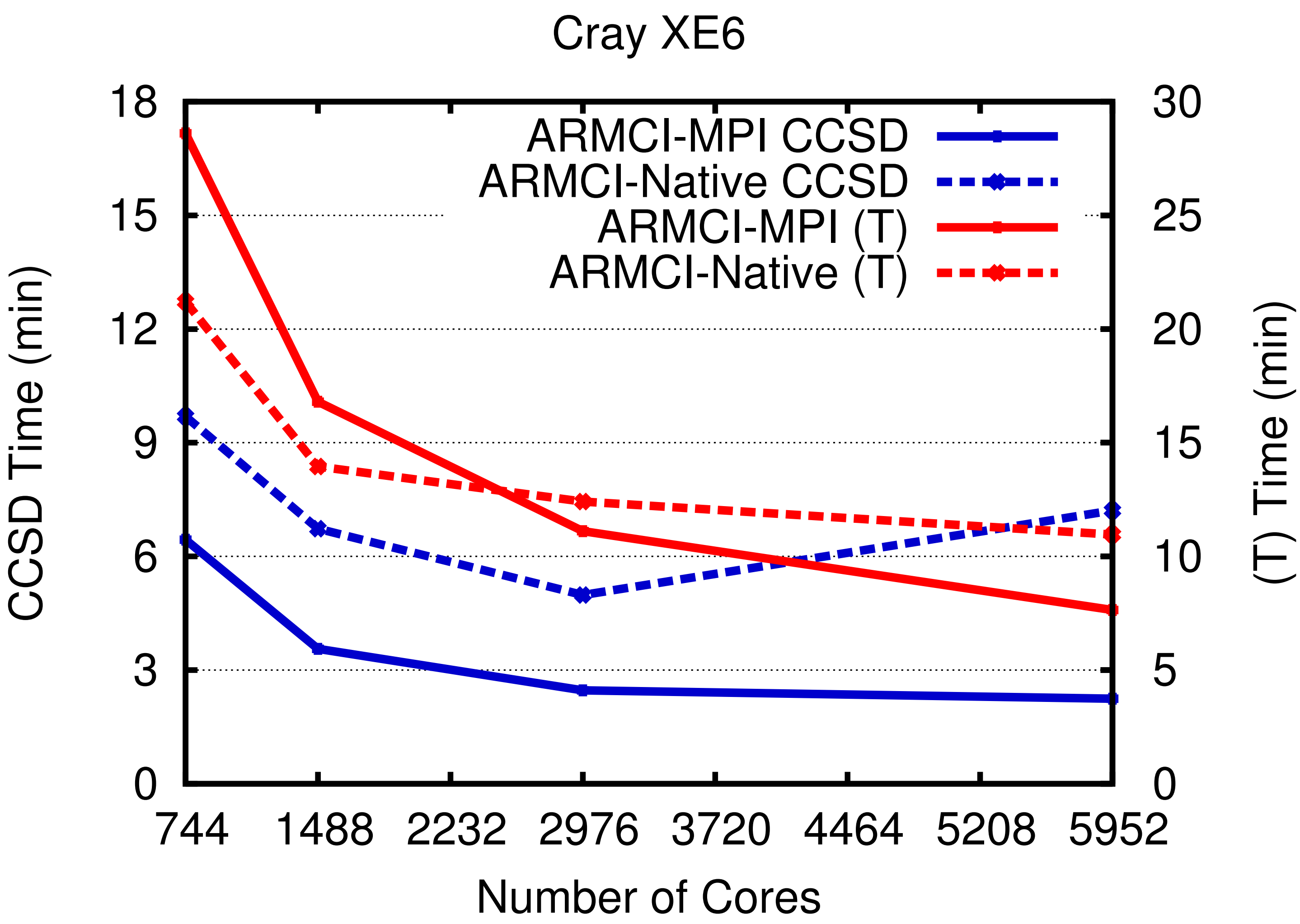
1. Shared data segment representation
   - MPI: <window, displacement>, window rank
   - ARMCI: <address>, absolute rank

   → Perform translation

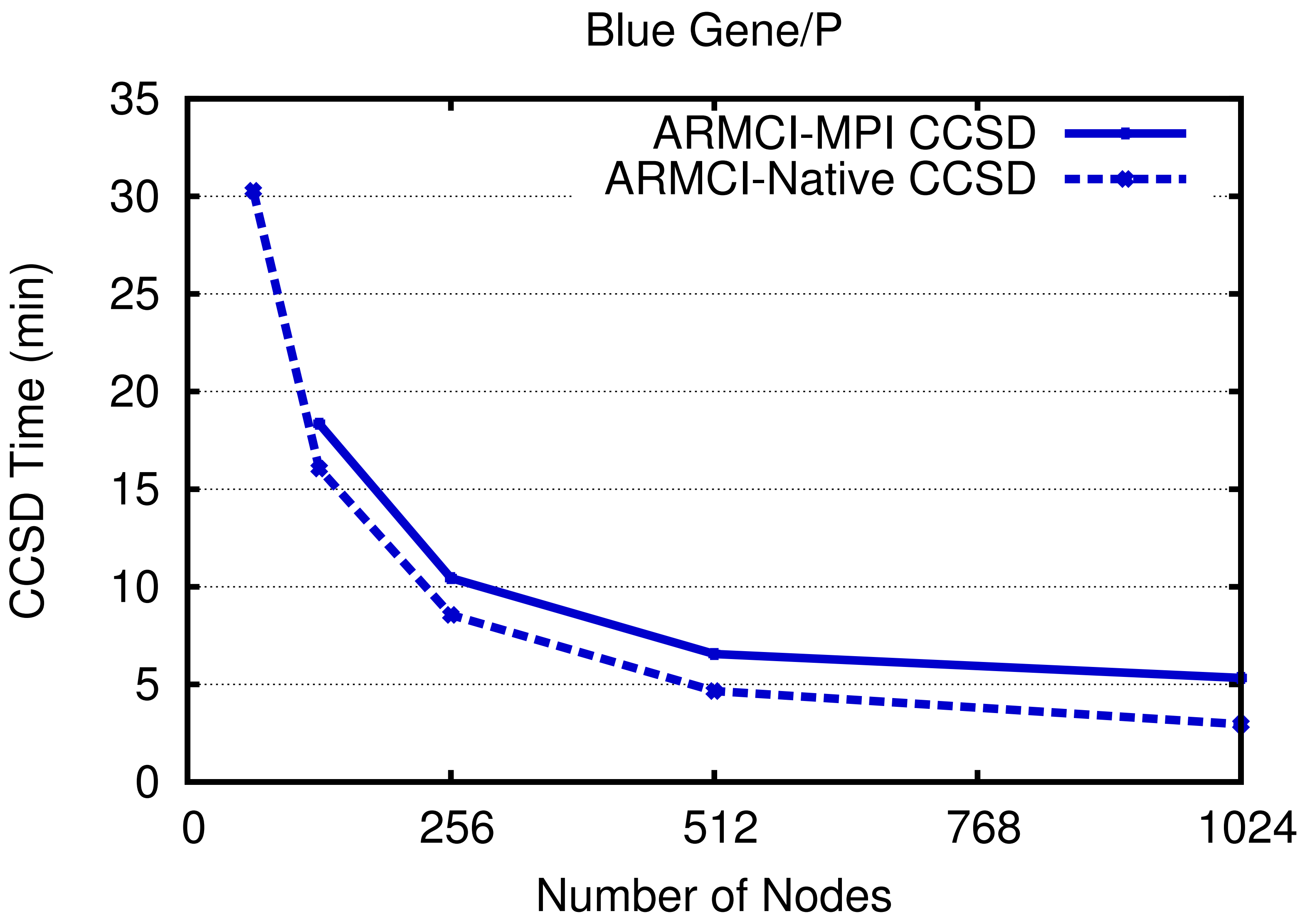
2. Data consistency model
   - ARMCI: Relaxed, location consistent for RMA, CCA undefined
   - MPI: Explicit (lock and unlock), CCA error

   → Explicitly maintain consistency, avoid CCA

NWChem Performance (XE6)

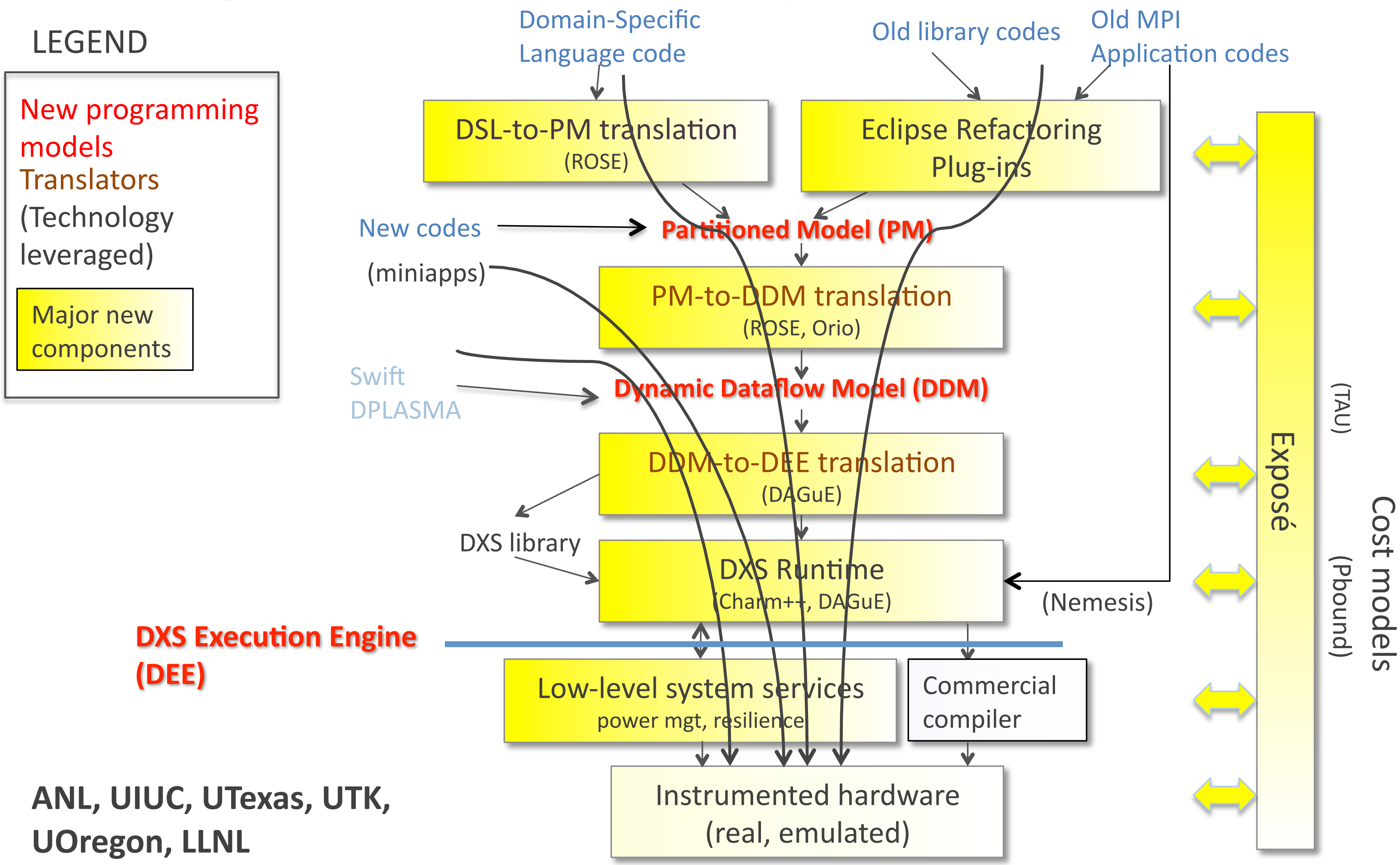


NWChem Performance (BG/P)



An idea and a proposal in need of funding

The Deep Exascale Stack (DXS)



ANL, UIUC, UTexas, UTK, UOregon, LLNL