Porting Charm++/NAMD to IBM Blue Gene/Q

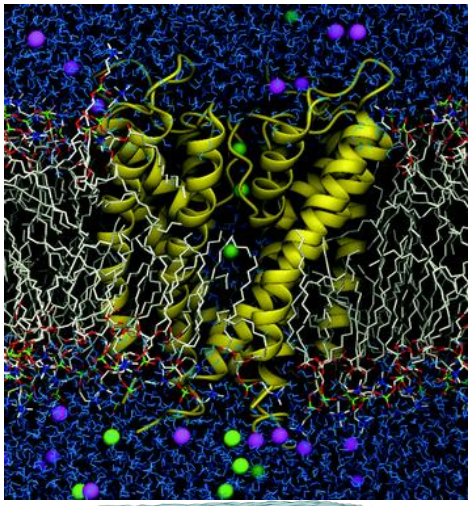
Wei Jiang
Argonne Leadership Computing Facility
7th, March

NAMD_esp

NAMD - Parallel molecular dynamics code designed for high-performance simulation of large biomolecular systems

Portable to all popular supercomputing platforms

Great scalability based on Charm++ parallel objects

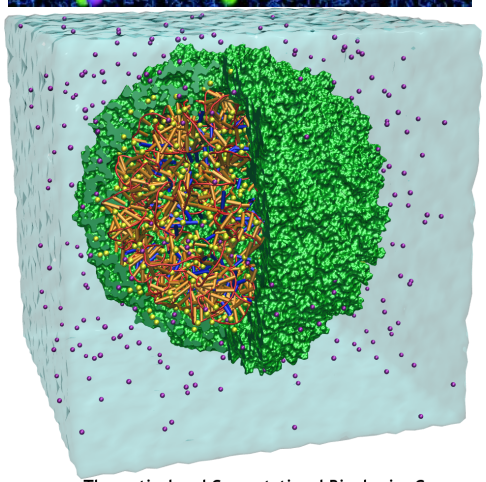


Scientific aims on Blue Gene/Q

Ensemble run that launches large number of replicas

concurrently - mainly for energetic simulation

High throughput simulation for large scale systems ~100M atoms



Theoretical and Computational Biophysics Group
Beckman Institute
University of Illinois at Urbana-Champaign

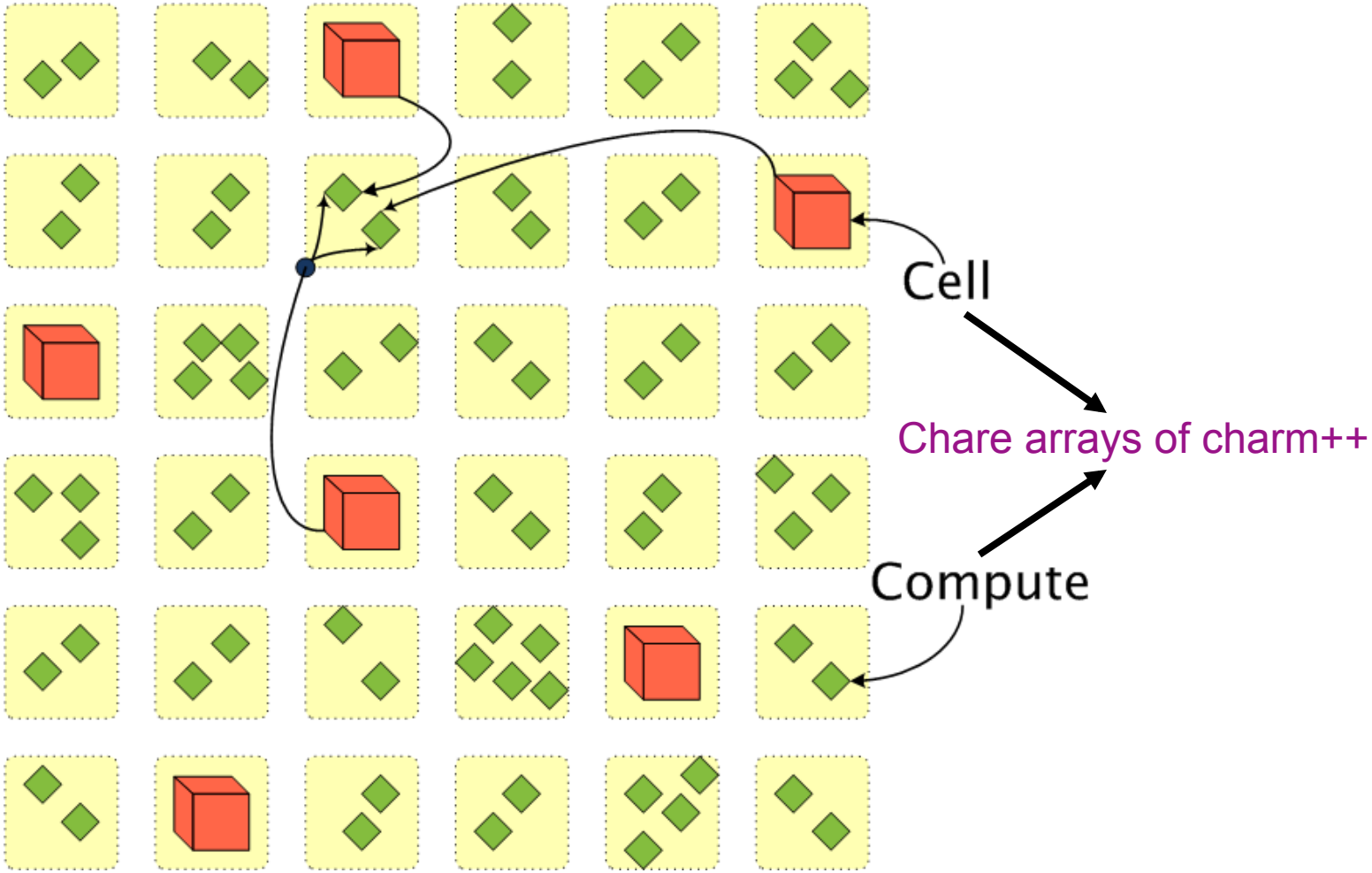
Requirements for charm++

New communication layer that supports parallel/parallel runs

Enable charm++ programming paradigm on Parallel Active Messaging Interface (PAMI)

Parallel Structure of NAMD

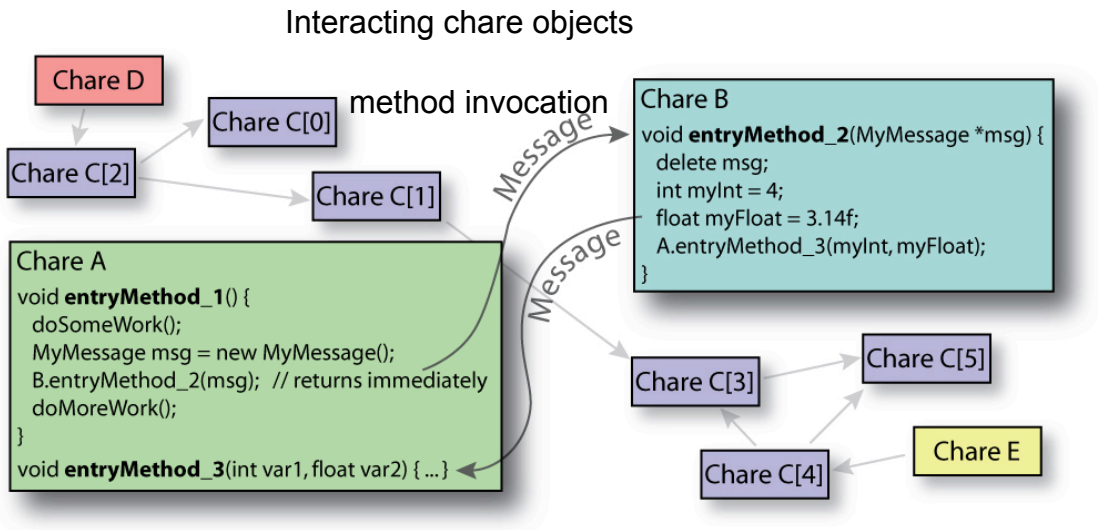
Hybrid force/spatial decomposition



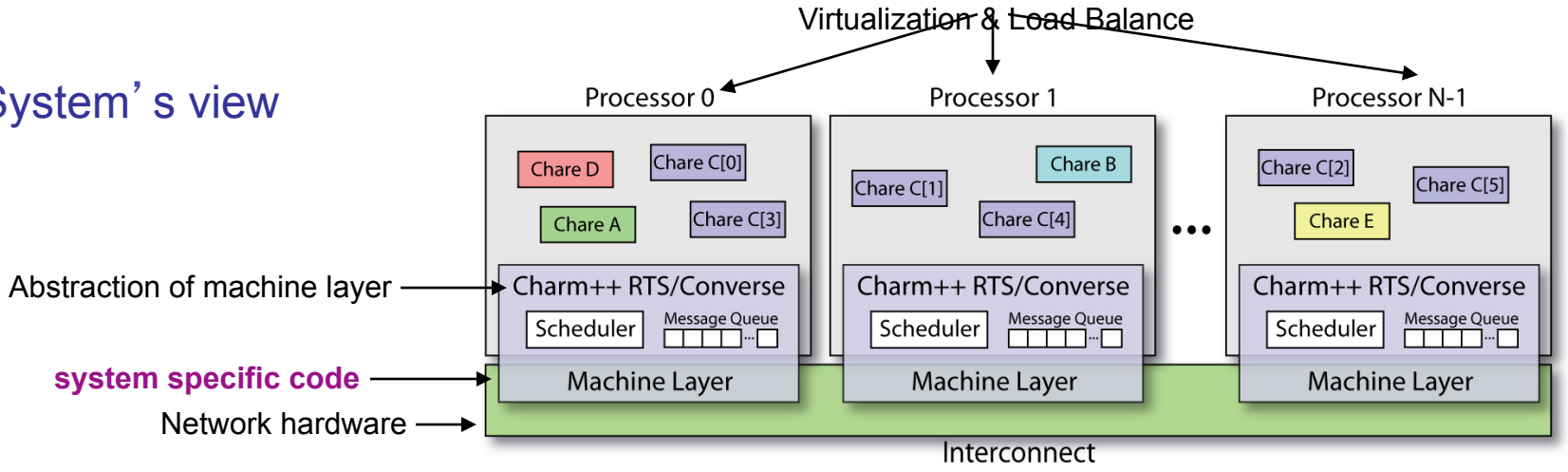
Adaptive Overlap of Communication and Computation
Dynamic Load Balancing

Charm++ runtime system

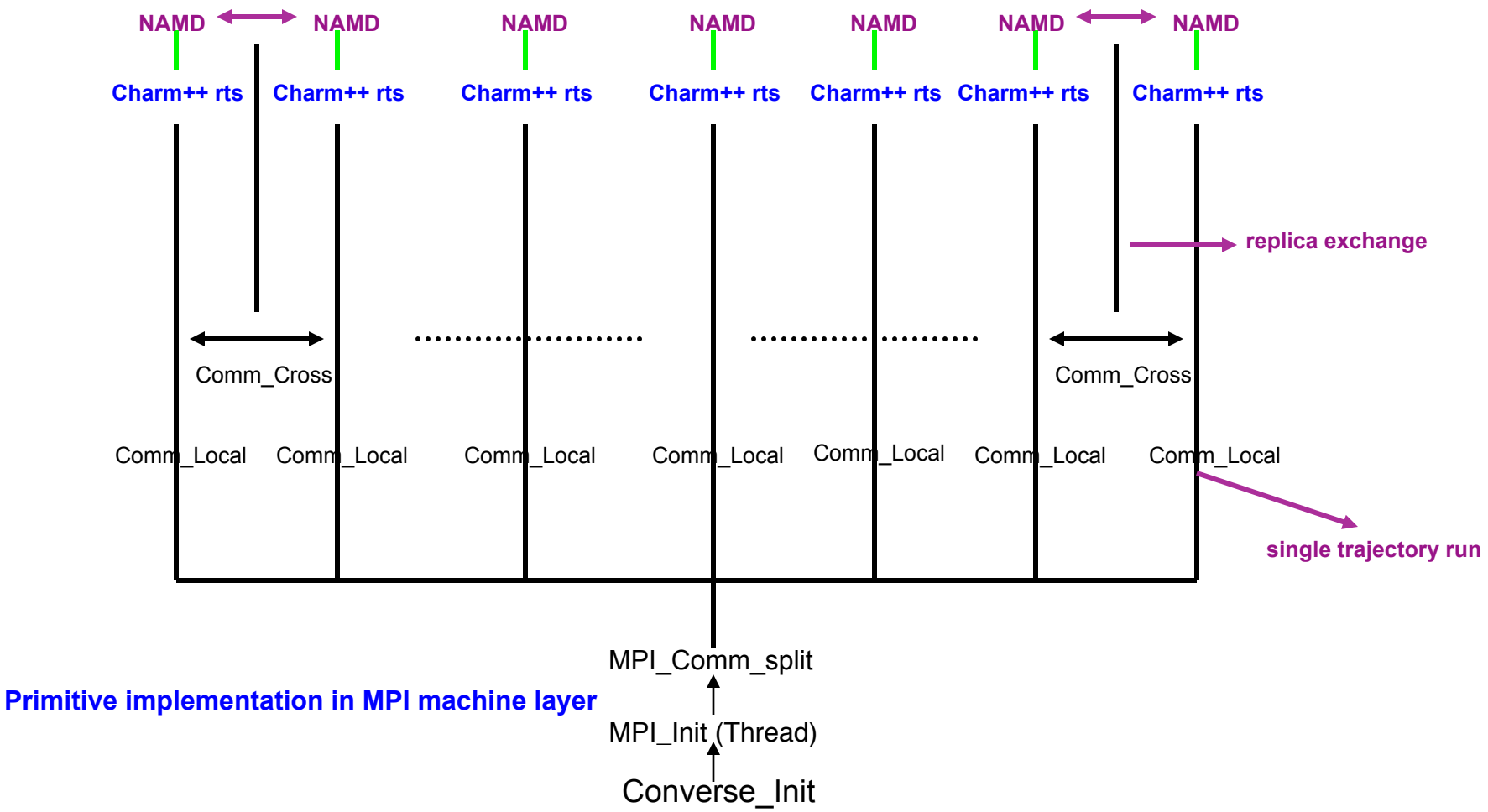
Application's view



System's view



Charm++ solution for NAMD ensemble run on BG/Q



Primitive implementation in MPI machine layer

Generic implementation in charm++ RTS/Converse layer

Mapping each global charm node onto local replicas for single trajectory run

Remapping each local charm node back to global one for replica-exchange

Ensemble run itself doesn't require optimization of low level communication library !

Porting charm++ to PAMI

Parallel Active Messaging Interface (PAMI)

- Multiple contexts accelerated by communication threads
- Client and context objects to support multiple programming paradigms
- Lockless algorithms to speed up message rate
- Novel techniques leveraging the new BG/Q architectural features
  - scalable atomic primitives implemented in the L2 cache
  - the highly parallel hardware messaging unit (MU)
  - the collective hardware acceleration

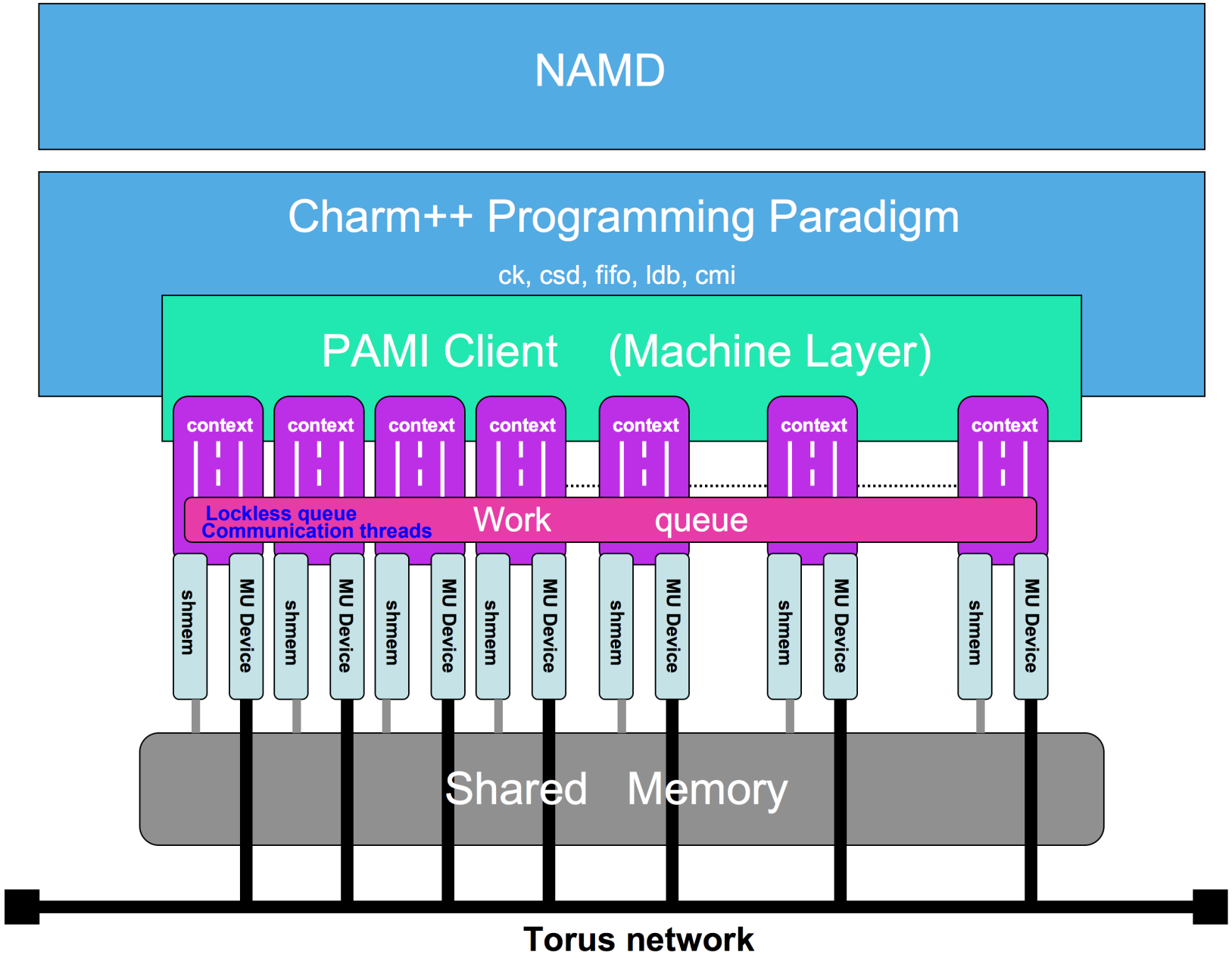
Non SMP charm++

- Traditional implementation and good performance

SMP charm++

- Multiple (Posix) threads run in the same process
- Minimize communication overheads
- Allows processing elements to access memory up to 16GB
- With and without dedicated communication threads

Fine-grained parallelism with multiple PAMI context objects



Lockless Queue & Scalable memory Allocator

Lockless queue

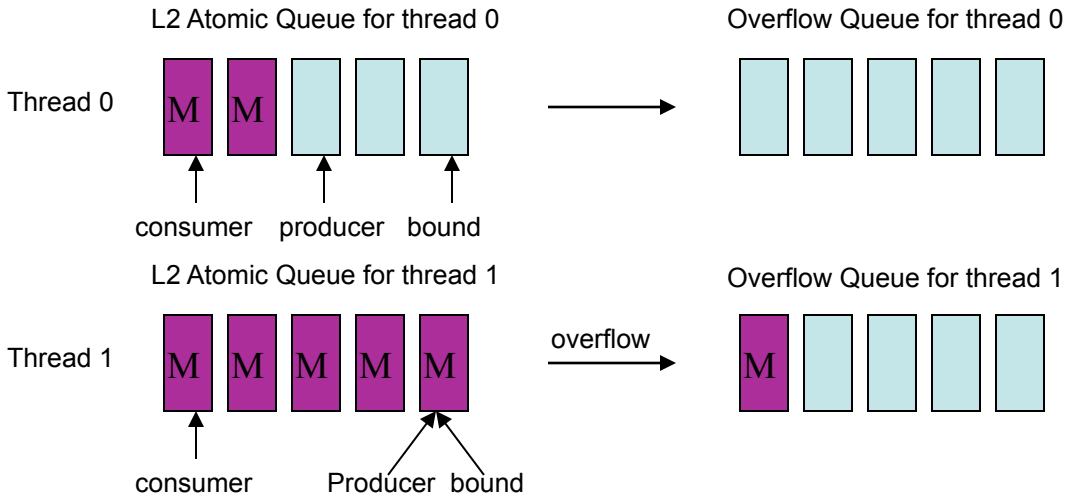
L2 atomic operations

Avoid mutex bottleneck

multiple threads simultaneously send messages to the same peer

Bounded load increment operation

Mutex-protected overflow queue



Scalable memory Allocation

GNU memory allocator: lock contention on multiple free calls

multiple threads received messages from the same peer

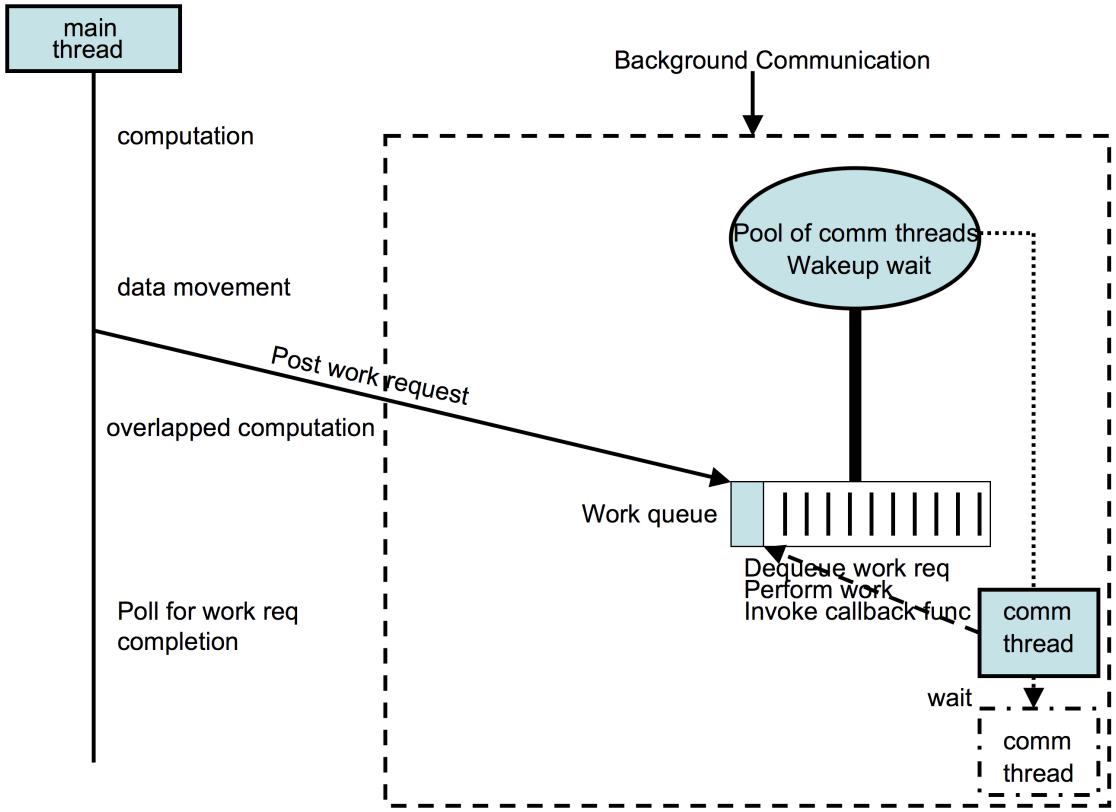
L2 atomic queue for each thread to store a pool of temporary buffers

A threshold for the memory pools after which buffers are freed to the memory heap

Exploiting Communication Threads

PAMI Communication threads

Perform **background** advance



Take advantage of the wakeup unit
Multiple communication threads can accelerate messages from several worker threads
Communication load evenly distributed across all the communication threads

Manytomany Interface

Motivation: Neighborhood Collectives

Each processor exchanges data with a different subset of ranks in a communicator

- Examples

– Boundry exchange
• Neighborhoodgather
– 3D FFT with pencil decomposition
• Neighborhoodalltoall
– Molecular dynamics real space communication
• Neighborhoodgatherandreduce

- No support in MPI 2 for neighborhood collectives

– Alltoallv, gatherv are typically not efficient with sparse neighborhoods

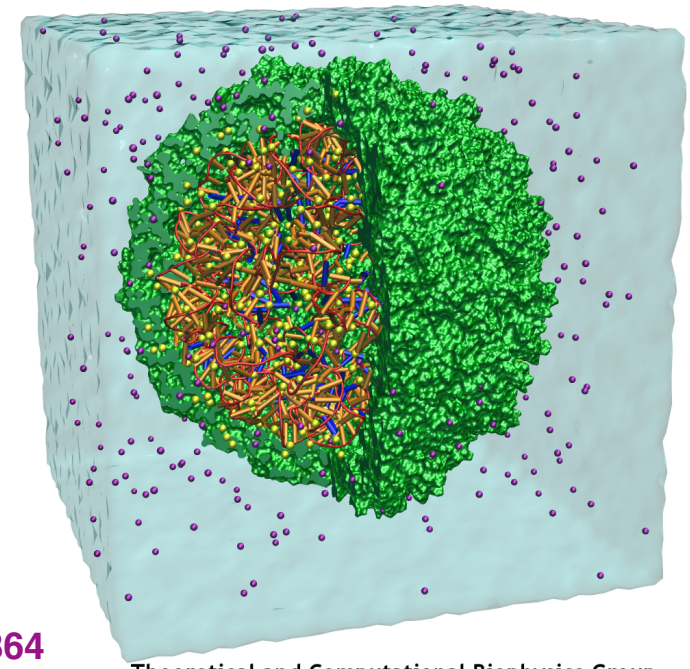
- **Performance killer** of molecular dynamics simulations

Converse Manytomany Interface

Chares send a burst of short messages to neighboring chares in a **single** optimized call
Messages are setup ahead of time and registered with a handle
Generates a list of sends/receives and completes them on multiple communication threads
Significantly accelerated message rate with multiple communication threads

NAMD Benchmarks

Multicontext,
Lockless queue
Scalable memory allocator
→ 1 proc/BGQ node



20M Satellite Tobacco Mosaic Virus (STMV)

Very large PME computation with a grid size of 216 × 1080 × 864 that limits scaling of the standard NAMD PME computation !

Nodes	Cores	Procs/node	Threads	Benchmark Performance (ms)	Options
1024	16384	1	64	38.4	
2048	32768	1	64	24.7	
4096	65536	1	48	15.9	comm threads, many-to-many
8192	131072	1	48	8.7	comm threads, many-to-many

100M STMV

PME grid size of 1080 × 1080 × 864

Nodes	Cores	Procs/node	Threads	Benchmark Performance (ms)	Options
2048	32768	1	48	98.8	comm threads, many-to-many
4096	65536	1	48	55.4	comm threads, many-to-many
8192	131072	1	48	30.3	comm threads, many-to-many
16384	262144	1	32	17.9	comm threads, many-to-many

Summary

Charm++ is efficiently ported to BGQ
Significant synergy between BG/Q software and Charm++
Novel fine-grained threading techniques
Generic solution for concurrent multiple copy run

Acknowledgement

Sameer Kumar (Leader of charm++ optimization on BGQ, IBM)

Parallel Programming Laboratory, UIUC

Jim Phillips (Beckman Institute, UIUC)

Ray Loy (ALCF, ANL)