A Parallel-Object Programming Model for PetaFLOPS Machines and BlueGene/Cyclops

Gengbin Zheng, Arun Singla,

Joshua Unger, Laxmikant Kalé

Parallel Programming Laboratory

Department of Computer Science

University of Illinois at Urbana-Champaign

http://charm.cs.uiuc.edu

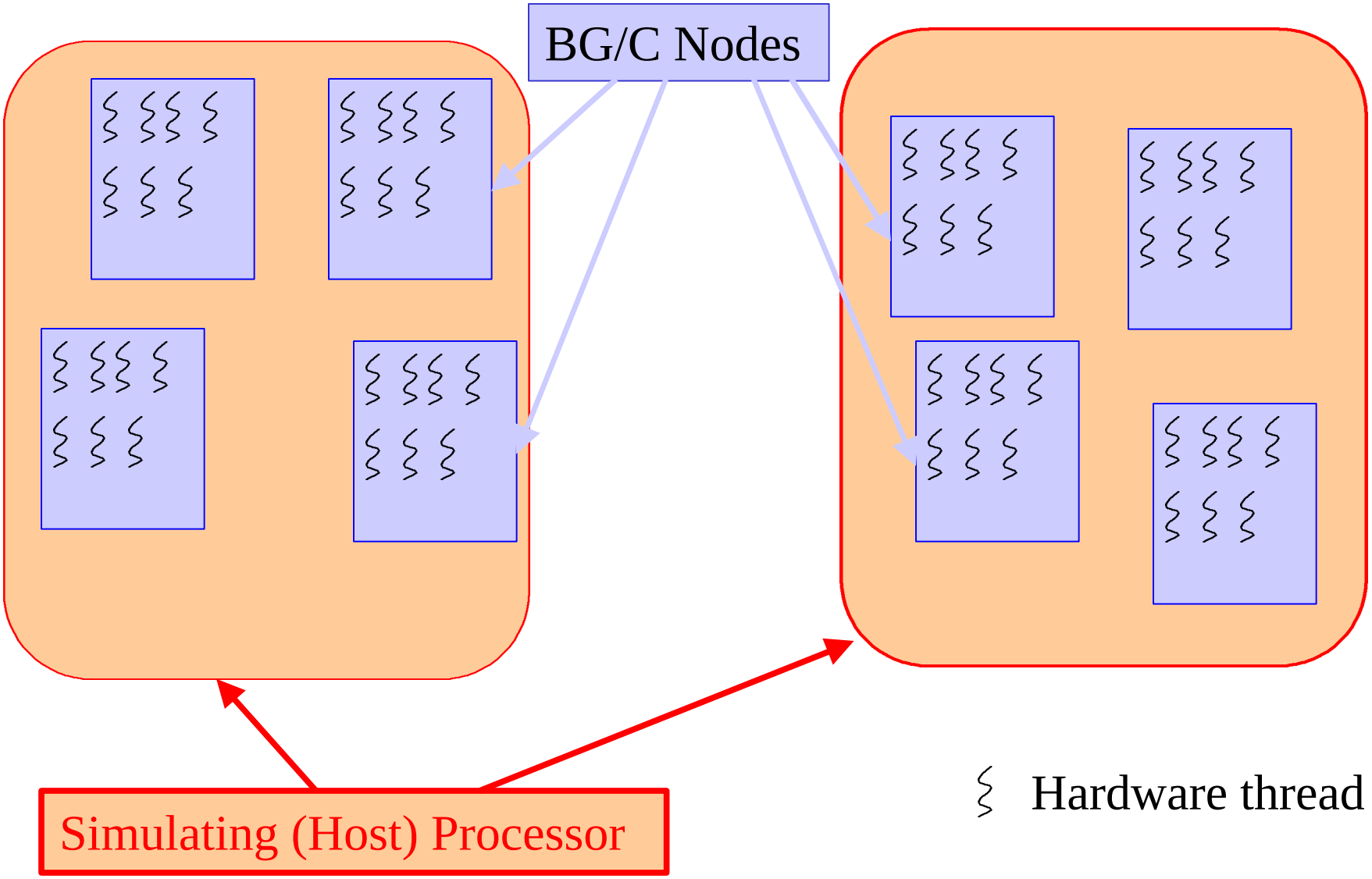
Massive Parallel Processors-In-Memory

- MPPIM
  - Large number of identical chips
  - Each contains multiple processors and memory
- Blue Gene/C
  - 34 x 34 x 36 cube
  - Multi-million hardware threads
- Challenges
  - How to program?
  - Software challenges: cost-effective

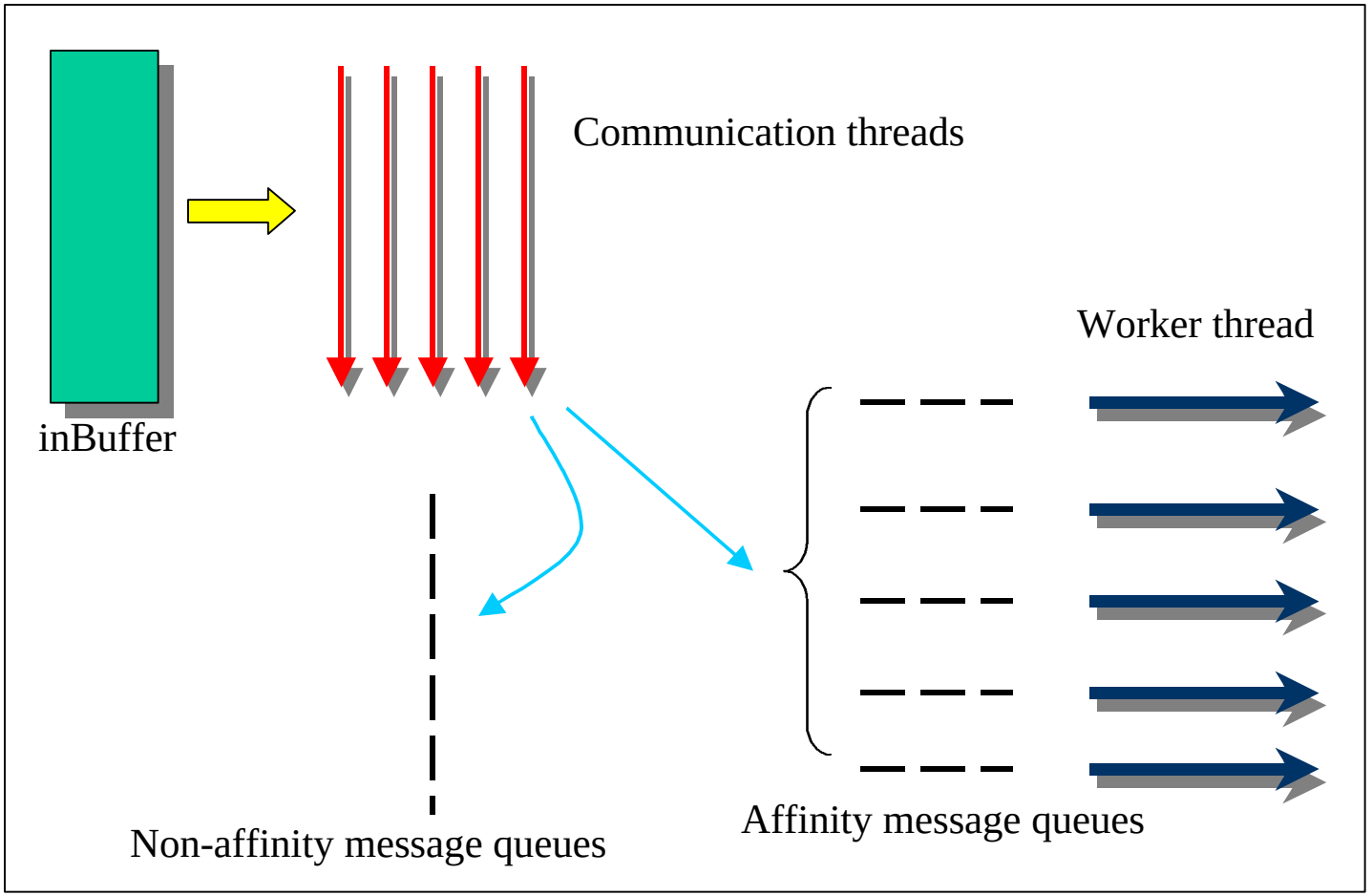
Need for Emulator

- Emulate BG/C machine API on conventional supercomputers and clusters.
  - Emulator enables programmer to develop, compile, and run software using programming interface that will be used in actual machine
- Performance estimation (with proper time stamping)
- Allow further research on high level parallel languages like Charm++
- Low memory-to-processor ratio make it possible
  - Half terabyte memory require 1000 processors 512MB

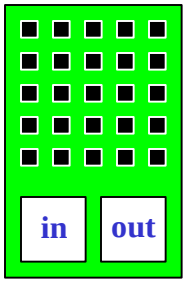
Emulation on a Parallel Machine



Bluegene Emulator
one BG/C Node



Blue Gene Programming API



- Low-level
  - Machine initialization
    - Get node ID: (x, y, z)
    - Get Blue Gene size
  - Register handler functions on node
  - Send packets to other nodes (x,y,z)
    - With handler ID

Blue Gene application example - Ring

```
typedef struct {
  char core[CmiBlueGeneMsgHeaderSizeBytes];
  int data;
} RingMsg;

void BgNodeStart(int argc, char **argv)  {
     int x,y,z, nx, ny, nz;
    RingMsg msg;                          msg.data =  888;
     BgGetXYZ(&x, &y, &z);           nextxyz(x, y, z, &nx, &ny, &nz);
     if (x == 0 && y==0 && z==0)
          BgSendPacket(nx, ny, nz, passRingID, LARGE_WORK, sizeof(int), (char *)&msg);
}

void passRing(char *msg)  {
     int x, y, z,  nx, ny, nz;
     BgGetXYZ(&x, &y, &z);           nextxyz(x, y, z, &nx, &ny, &nz);
     if (x==0 && y==0 && z==0)     if (++iter == MAXITER) BgShutdown();
     BgSendPacket(nx, ny, nz, passRingID, LARGE_WORK, sizeof(int), msg);
}
```

Emulator Status

- Implemented on Charm++/Converse
  - 8 Million processors being emulated on 100 ASCI-Red processors
- How much time does it take to run an emulation v.s. how much time does it take to run on real BG/C?
  - Timestamp module
- Emulation efficiency
  - On a Linux cluster:
    - Emulation shows good speedup(later slides)

Programming issues for MPPIM

- Need higher level of programming language
- Data locality
- Parallelism
- Load balancing
- Charm++ is a good programming model candidate for MPPIMs

Charm++

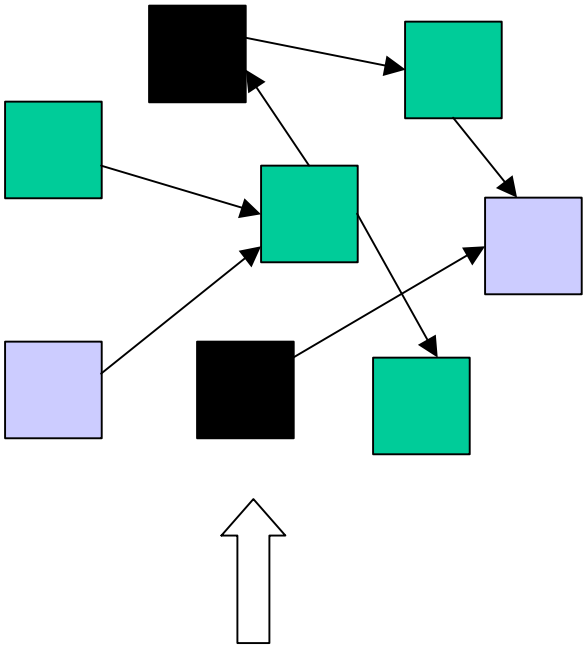
- Parallel C++ with *Data Driven* Objects
- Object Arrays/ Object Collections
- Object Groups:
  - Global object with a “representative” on each PE
- Asynchronous method invocation
- Built-in load balancing(runtime)
- Mature, robust, portable
- http://charm.cs.uiuc.edu

Multi-partition Decomposition

- Idea: divide the computation into a large number of pieces(parallel objects)
  - Independent of number of processors
  - Typically larger than number of processors
  - Let the system map entities to processors
- Optimal division of labor between “system” and programmer:
    - Decomposition done by programmer,
    - Everything else automated

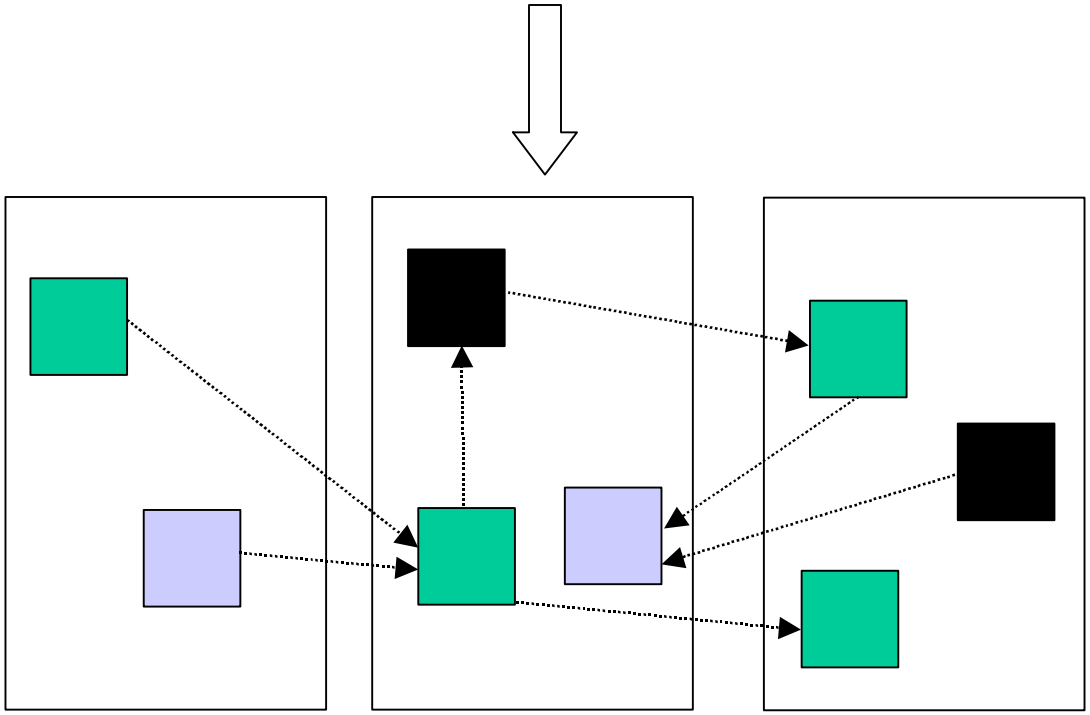
Object-based Parallelization

User is only concerned with interaction between objects



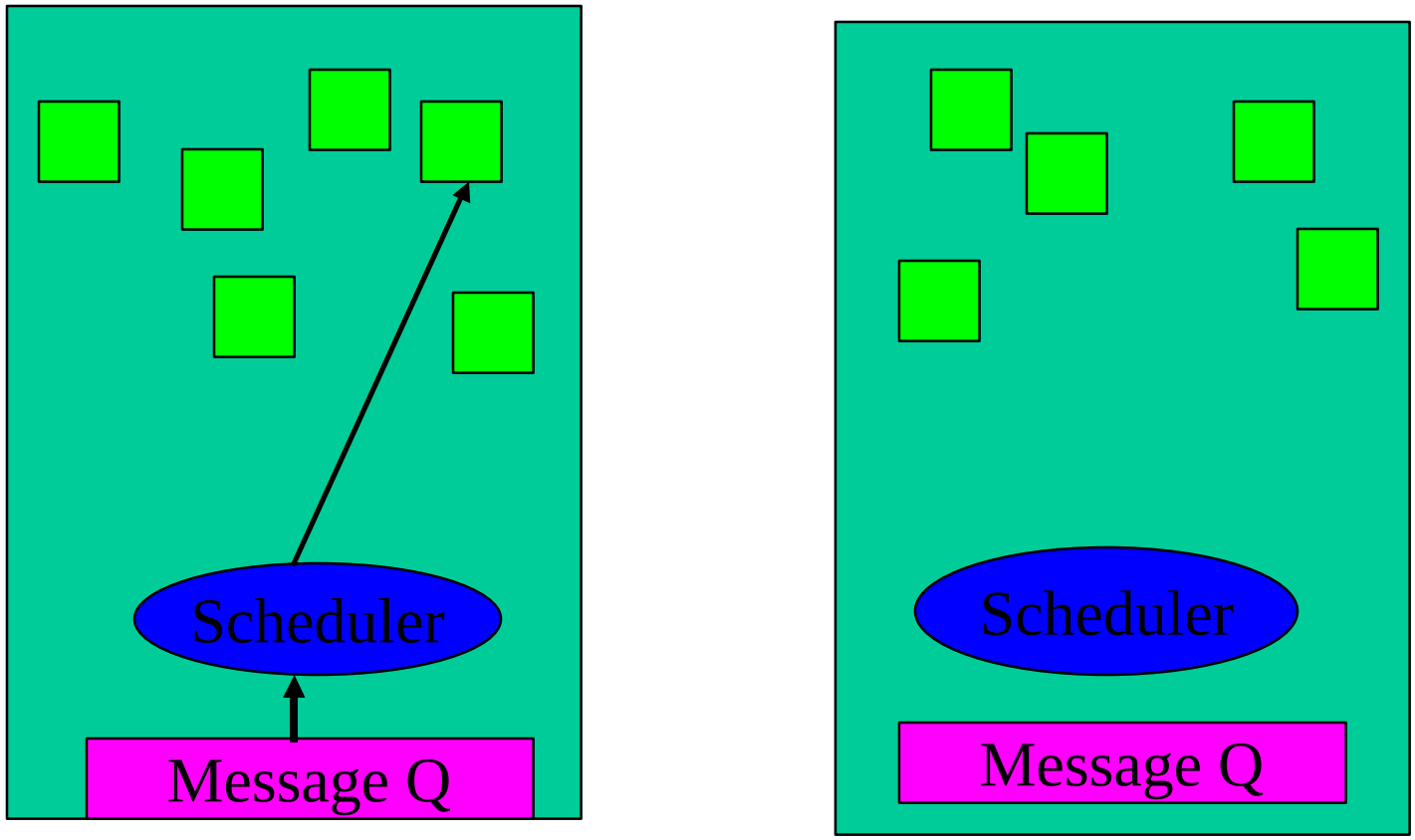
User View

System implementation



Charm++ PE

Data driven execution



Load Balancing Framework

- Based on object migration
  - Partitions implemented as objects (or threads) are mapped to available processors by LB framework
- Measurement based load balancers:
  - Principle of persistence
    - Computational loads and communication patterns
  - Runtime system measures actual computation times of every partition, as well as communication patterns
- Variety of “plug-in” LB strategies available
  - Scalable to a few thousand processors
  - Including those for situations when principle of persistence does not apply

Charm++ is a Good Match for MPPIM

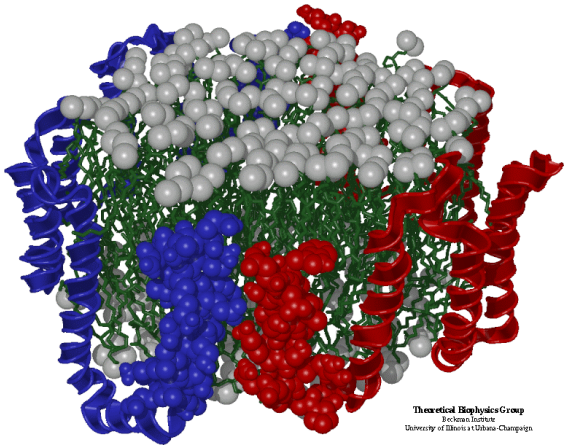
- Message driven/Data driven
- Encapsulation : objects
- Explicit cost model:
  - Object data, read-only data, remote data
  - Aware of the cost of accessing remote data
- Migration and resource management: automatic
- One sided communication
- Asynchronous global operations (reductions, ..)

Charm++ Applications

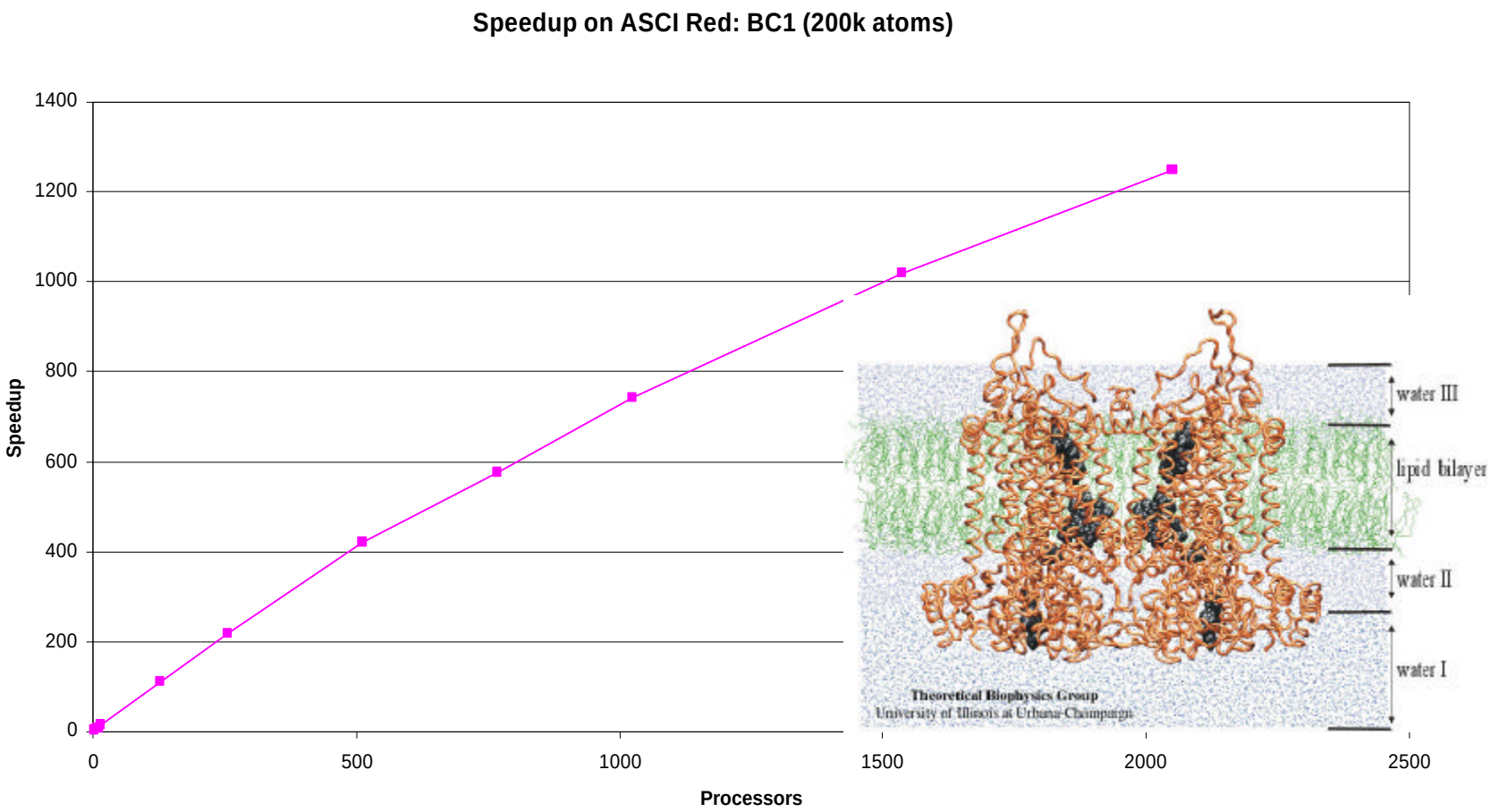
- Charm++ developed in the context of real applications
- Current applications we are involved with:
  - Molecular dynamics(NAMD)
  - Crack propagation
  - Rocket simulation: fluid dynamics + structures +
  - QM/MM: Material properties via quantum mech
  - Cosmology simulations: parallel analysis+viz
  - Cosmology: gravitational with multiple timestepping

Molecular Dynamics

- Collection of [charged] atoms, with bonds
- Newtonian mechanics
- At each time-step
  - Calculate forces on each atom
    - Bonds:
    - Non-bonded: electrostatic and van der Waal's
  - Calculate velocities and advance positions
- 1 femtosecond time-step, millions needed!
- Thousands of atoms (1,000 - 100,000)

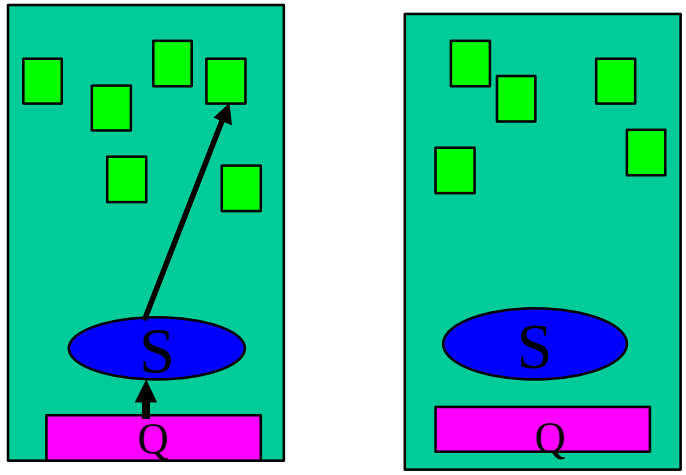


Performance Data: SC2000



Further Match With MPPIM

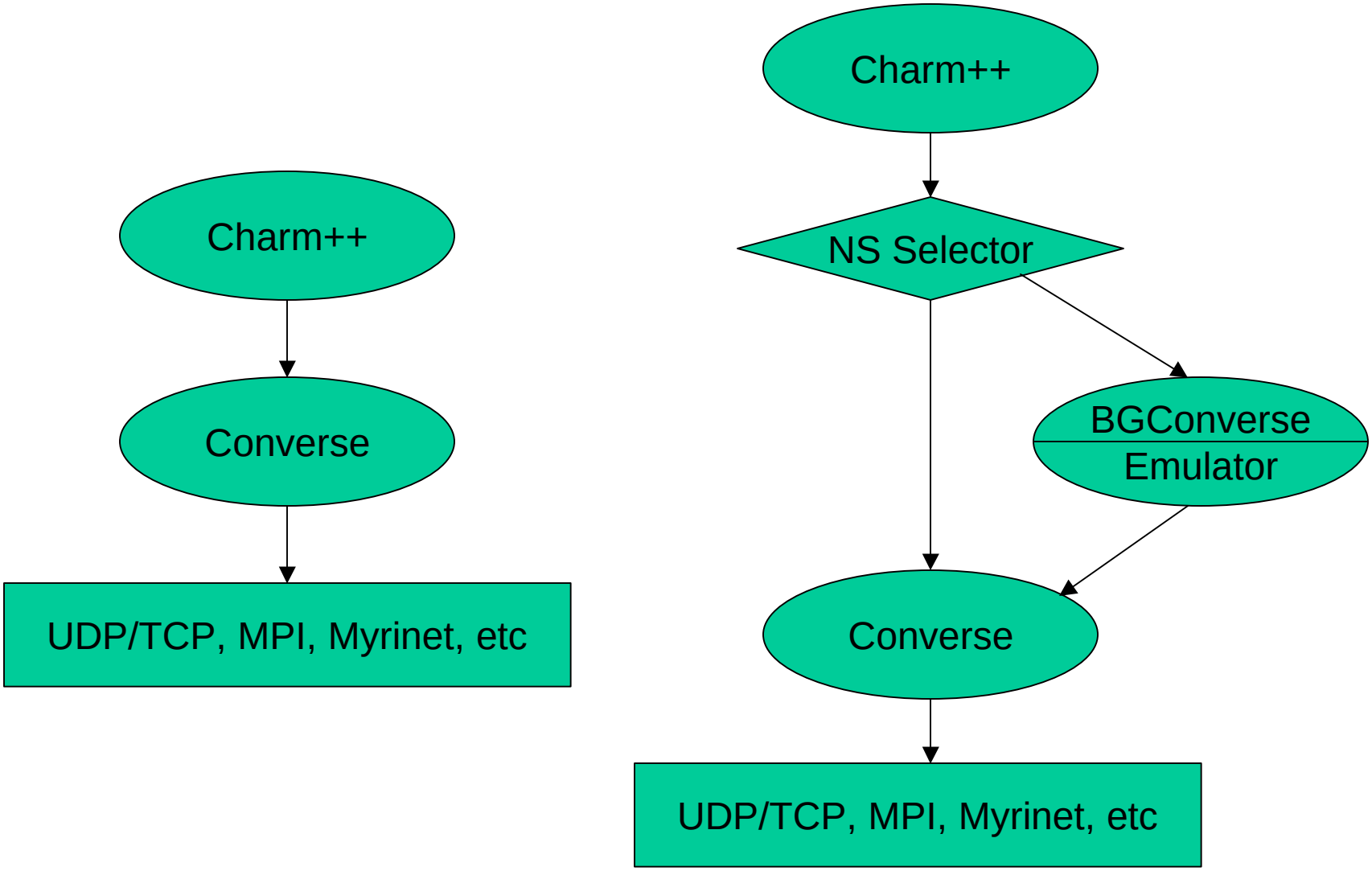
- Ability to predict:
  - Which data is going to be needed and which code will execute
  - Based on the ready queue of object method invocations
  - So, we can:
    - Prefetch data accurately
    - Prefetch code if needed



Blue Gene/C Charm++

- Implemented Charm++ on Blue Gene/C Emulator
  - Almost all existing Charm++ applications can run w/o change on emulator
- Case study on some real applications
  - leanMD: Fully functional MD with only cutoff (PME later)
  - AMR
- Time stamping(ongoing work)
  - Log generation and correction

Parallel Object Programming Model



BG/C Charm++

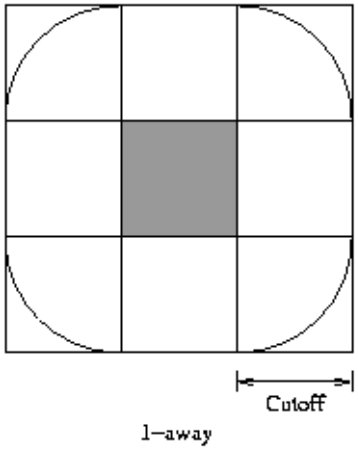
- Object affinity
  - Object mapped to a BG node
    - A message can be executed by any thread
    - Load balancing at node level
    - Locking needed
  - Object mapped to a BG thread
    - An object is created on a particular thread
    - All messages to the object will go to that thread
    - No locking needed.
    - Load balancing at thread level

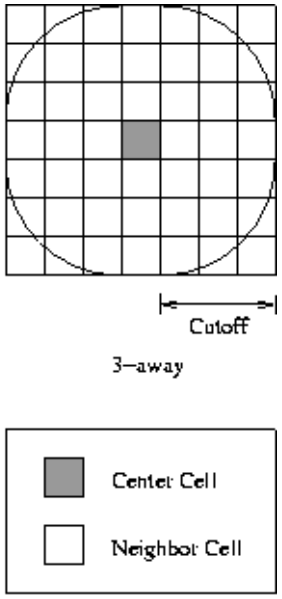
Applications on the current system

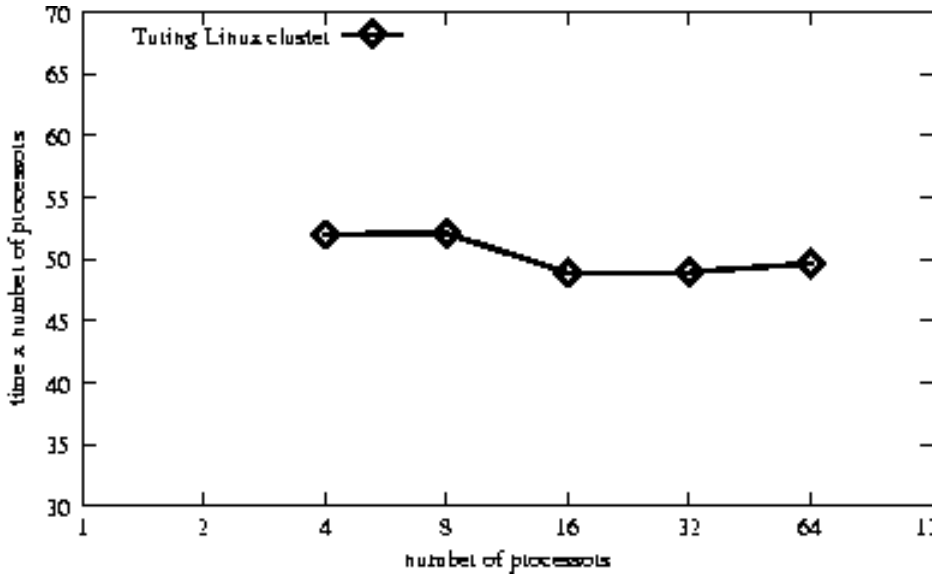
- LeanMD:
  - Research quality Molecular Dynamics
  - Version 0: only electrostatics + van der Vaal
- Simple AMR kernel
  - Adaptive tree to generate millions of objects
    - Each holding a 3D array
  - Communication with “neighbors”
    - Tree makes it harder to find nbrs, but Charm makes it easy

LeanMD

- K-array molecular dynamics simulation
- Using Charm++ Chare arrays







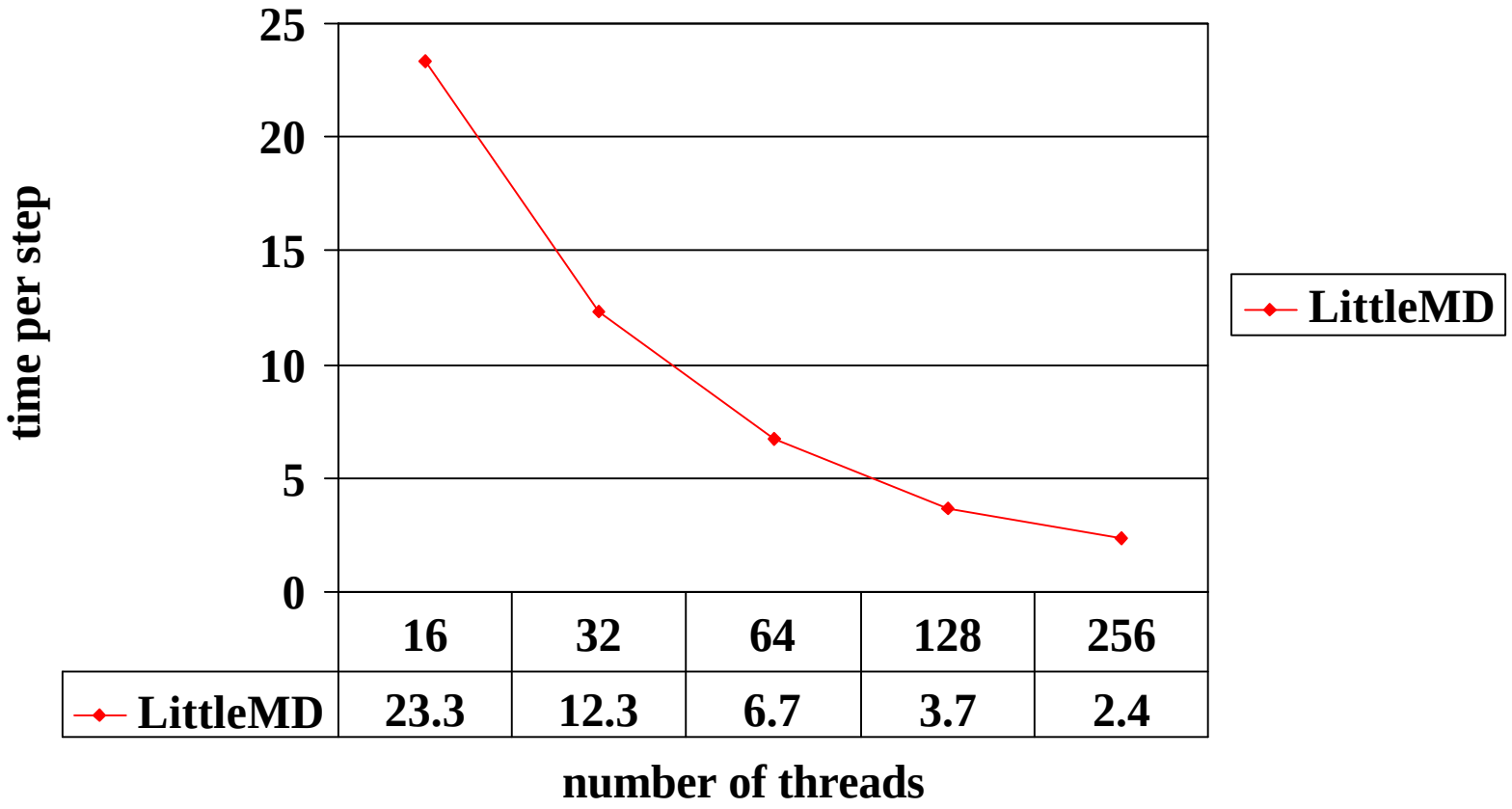
- 10x10x10 200 threads each
- 11x11x11 cells
- 144914 cell-to-cell computes

Correction of Time stamps at runtime back

- Timestamp
  - Per thread timer
  - Message arrive time
    - Calculate at time of sending
      - Based on hop and corner
    - Update thread timer when arrive
- Correction needed for out-of-order messages
  - Correction messages send out


File
Tools
Hel
0000
6450000
6460000
6470000
6480000
6490000
6500000
6510
Processor 0
(58.672%)
Processor 1
(61.659%)
Processor 2
(66.936%)
Processor 3
(71.658%)
Processor 4
(72.634%)
Processor 5
(72.611%)
Processor 6
(76.076%)
Processor 7
(78.284%)
Processor 8
(77.548%)
Processor 9
(78.385%)
Processor 10
(81.448%)
Processor 11
(79.358%)
Processor 12
(78.35%)
Processor 13
CentralLB::ReceiveMigration(LBMigrateMsg* impl_msg)
Msg Len: 12396
Begin Time: 6462180
End Time: 6466999
Total Time: 4.819ms (0.41493776%)
Packing: 0.020ms
Msgs created: 15
Created by processor 0
Display Pack Times
Display Message Creations
Display Idle Time
Select Ranges
Change Colors
<<
SCALE:
2
>>
Reset

LittleMD Blue Gene Time



- 200,000 atoms
- Use 4 simulating processors

Summary

- Emulation of BG/C with millions of threads
  - On conventional supercomputers and clusters
- Charm++ on BG Emulator
  - Legacy Charm++ applications
  - Load balancing(need more research)
- We have Implemented multi-million object applications using Charm++
  - And tested on emulated Blue Gene/C
- Getting accurate simulating timing data
- More info: http://charm.cs.uiuc.edu
  - Both Emulator and BG Charm++ are available for download