LRTS: A Portable High Performance Low-level Communication Interface

Yanhua Sun[1] Laxmikant(Sanjay) V. Kále[1]

[1]University of Illinois at Urbana-Champaign

sun51@illinois.edu

April 15, 2013

Motivation

- What the vendors provide
  - Modern supercomputers, especially networks, are complicated

Motivation

- What the vendors provide
  - Modern supercomputers, especially networks, are complicated
- What the programming models require
  - Global address space models
  - Message passing model
  - Message driven (active message) models

Motivation

- What the vendors provide
  - Modern supercomputers, especially networks, are complicated
- What the programming models require
  - Global address space models
  - Message passing model
  - Message driven (active message) models
- A minimum set of functions to implement runtime systems

Outline

- Goal of LRTS
- Charm++ architecture on LRTS
- Core APIs and extended APIs
- Performance of micro benchmarks and NAMD
- Future work

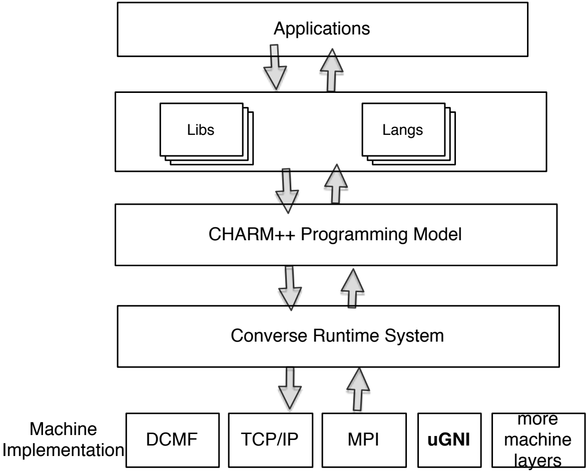
Goals of LRTS

Goal
$=$ Completeness $+$ Productivity $+$ Portability $+$ Performance

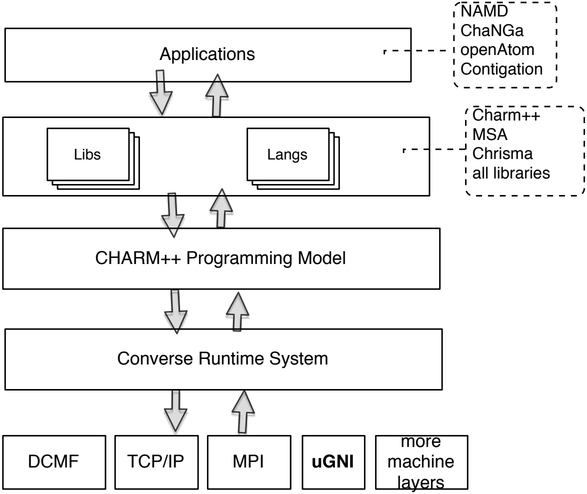
Goal of LRTS

- Completeness
  - Sufficient to run Charm++
- Productivity
  - Require no knowledge of Charm++ to port
  - Charm++ developers : easy to add new features (Replica)
- Portability
  - Functions should not dependend on specific machines
- Performance
  - Space for optimization

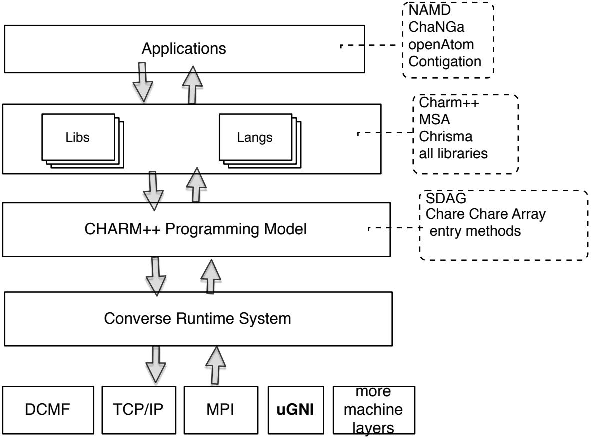
Charm++ Architecture



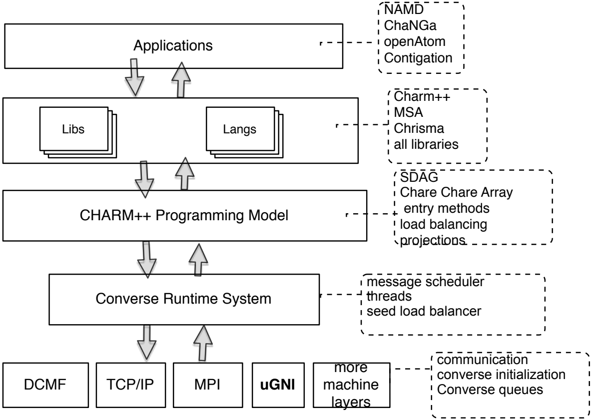
Charm++ Architecture



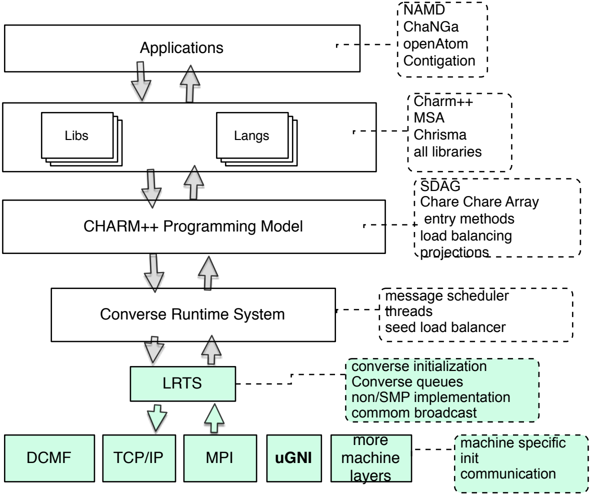
Charm++ Architecture



Charm++ Architecture



Charm++ Architecture Based on LRTS



Charm++ Naming Rules

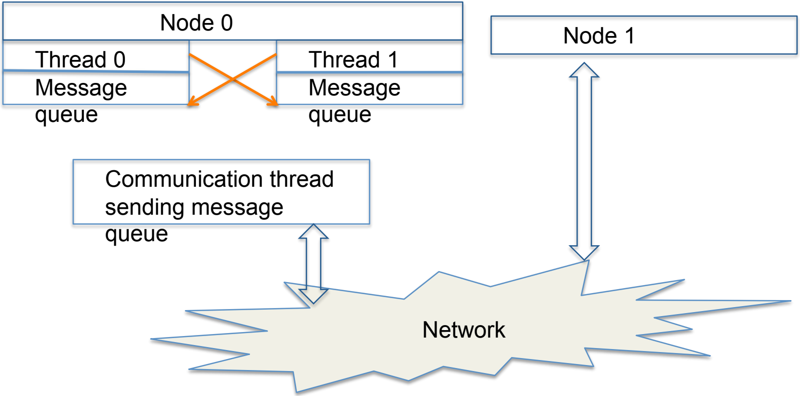
- CkFoo (most used for Charm++ programmers)
- CmiFoo (converse programs)
- LrtsFoo (only for vendors)

Messaging Flow

- Non SMP mode - one process per core (hardware thread)
- SMP mode - one thread per core (hardware thread)
  - Intra-node communication by passing pointers
  - Dedicated communication thread

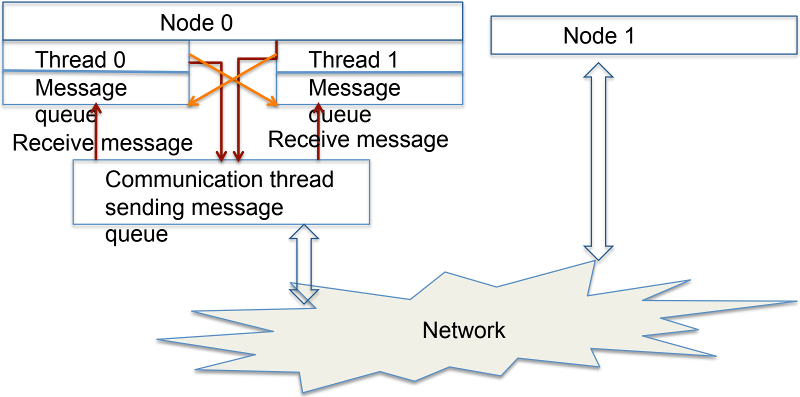
Messaging Flow

- Non SMP mode - one process per core (hardware thread)
- SMP mode - one thread per core (hardware thread)
  - Intra-node communication by passing pointers
  - Dedicated communication thread



Messaging Flow

- non SMP mode - one process per core (hardware thread)
- SMP mode - one thread per core (hardware thread)
  - Intra-node communication by passing pointers
  - Dedicated communication thread



Core APIs

required to run Charm++

Startup and Shutdown

- **void LrtsInit(int *argc, char ***argv, int *numNodes, int *myNodeID)**
- **void LrtsExit()**
- **void LrtsBarrier()**

Core APIs - P2P communication

Sending messages

CmiCommHandle LrtsSendFunc(int destNode, int destPE, int size, char *msg, int mode);

- Different protocols for message size
- Buffering scheme in machine layer

Core APIs - P2P communication

Sending messages

CmiCommHandle LrtsSendFunc(int destNode, int destPE, int size, char *msg, int mode);

- Different protocols for message size
- Buffering scheme in machine layer

LrtsAdvanceCommunication

void LrtsAdvanceCommunication(int whileidle);

- Sending buffered messages
- Polling network

Core APIs - P2P communication

Sending messages

CmiCommHandle LrtsSendFunc(int destNode, int destPE, int size, char *msg, int mode);

- Different protocols for message size
- Buffering scheme in machine layer

LrtsAdvanceCommunication

void LrtsAdvanceCommunication(int whileidle);

- Sending buffered messages
- Polling network
    - **void handleOneRecvedMsg(int size, char *msg)**

Extended APIs - Memory

Memory Management

void* LrtsAlloc(int n_bytes)
void LrtsFree(void *msg)

- Pinned memory pool - uGNI
- L2Atomic queues for freed messages

Extended APIs - Persistent Messages

Persistent messages

Communication partners and sizes do not change

Extended APIs - Persistent Messages

Persistent messages

Communication partners and sizes do not change

- RDMA support (uGNI, PAMI, Ibverbs)
- **void LrtsSendPersistentMsg(PersistentHandle h, int destNode, int size, void *msg)**

Extended APIs - Collectives

- **void LrtsBroadcast()**
  common implementation + specific

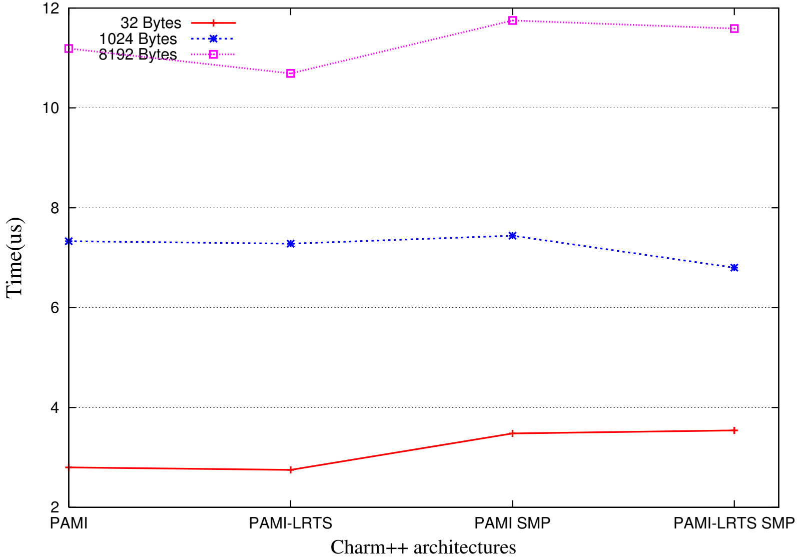
Extended APIs - Collectives

- **void LrtsBroadcast()**
  common implementation + specific
  - Spanning Tree
  - Hypercube
- All asynchronous functions

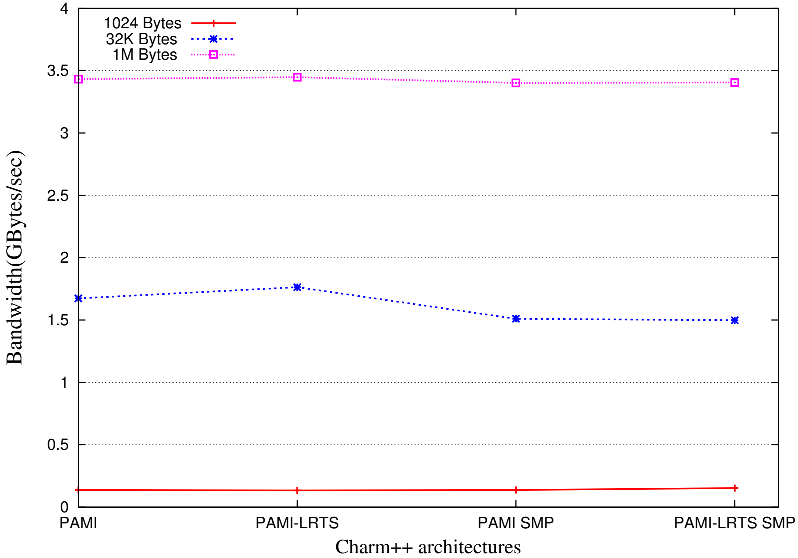
Status of LRTS

- Cray machines with uGNI : XE, XK, XC
  - Sun etal, A uGNI-Based Asynchronous Message-driven Runtime System for Cray Supercomputers with Gemini Interconnect, IPDPS 2012
  - Sun etal, Optimizing Fine-grained Communication in a Biomolecular Simulation Application on Cray XK6, SC 2012
- IBM machines : BlueGene/P with DCMF; BlueGene/Q with PAMI
  - Kumar etal, Acceleration of an Asynchronous Message Driven Programming Paradigm on IBM Blue Gene/Q, IPDPS 2013
- Machines supporting MPI
- Infiniband clusters

Performance - Latency on BGQ

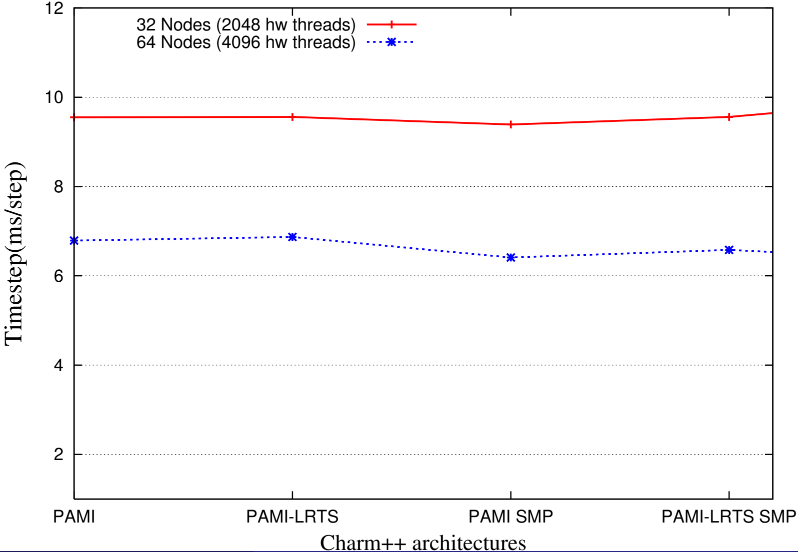


Performance - Bandwidth on BGQ



Application Performance

NAMD Apoa1(92k atoms) with PME every 4 steps on BGQ



100M-atom Simulation on State-of-art Machines

- Best performance on Blue Waters is 8.9ms/step with 25k nodes
- 13ms/step on Titan with 18k nodes
- 17.9ms/step on Bluegene/Q with 16K nodes

Conclusion and Future work

Conclusion

- LRTS interface simplifies the runtime implementation on new hardware
- LRTS maintain good performance

Conclusion and Future work

Conclusion

- LRTS interface simplifies the runtime implementation on new hardware
- LRTS maintain good performance

Future work

- Message buffering and scheduling
- Fault tolerance interface
- Implement other runtime system - Unistack