LRTS: A Portable High Performance Low-level Communication Interface

Yanhua Sun¹ Laxmikant(Sanjay) V. Kále¹

¹University 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 depend on specific machines
- Performance
  - Space for optimization
Charm++ Architecture

Applications

Libs
Langs

CHARM++ Programming Model

Converse Runtime System

DCMF TCP/IP MPI uGNI more machine layers

NAMD ChaNGa openAtom Contigation

Charm++ MSA Chrisma all libraries
Charm++ Architecture

- Applications
  - Libs
  - Langs
- CHARM++ Programming Model
  - NAMD
  - ChaNGa
  - openAtom
  - Contigation
- Converse Runtime System
  - Charm++
  - MSA
  - Chrisma
  - all libraries
  - SDAG
  - Share
  - Chare
  - Array
  - entry methods
  - load balancing
  - projections
- Converse Queues
  - message scheduler
  - threads
  - seed load balancer
- DCMF
- TCP/IP
- MPI
- uGNI
- more machine layers
- communication
  - converse initialization
  - Converse queues

Yanhua Sun  U of Illinois at Urbana-Champaign  9/24
Charm++ Architecture Based on LRTS

Applications

Libs

Languages

CHARM++ Programming Model

Converse Runtime System

DCMF
TCP/IP
MPI
uGNI

more machine layers

NAMD
ChaNGa
openAtom
Contigation

Charm++
MSA
Chrisma
all libraries

SDAG
Chare Chare Array
entry methods
load balancing
projections

message scheduler
threads
seed load balancer

converse initialization
Converse queues
non/SMP implementation
common broadcast

machine specific
init
communication
Charm++ Naming Rules

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

**Diagram:**

- Node 0
  - Thread 0
  - Message queue
  - Receive message
  - Communication thread sending message queue
- Node 1
  - Thread 1
  - Message queue
  - Receive message

**Network**
Core APIs

required to run Charm++

<table>
<thead>
<tr>
<th>Startup and Shutdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>void LrtsInit(int *argc, char ***argv, int *numNodes, int *myNodeID)</td>
</tr>
<tr>
<td>void LrtsExit()</td>
</tr>
<tr>
<td>void LrtsBarrier()</td>
</tr>
</tbody>
</table>
Core APIs - P2P communication

Sending messages

\texttt{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

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

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

### LrtsAdvanceCommunication

```c
void LrtsAdvanceCommunication(int whileidle);
```

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

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

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

Communication partners and sizes do not change
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
void LrtsBroadcast()
common implementation + specific
  - Spanning Tree
  - Hypercube

All asynchronous functions
Cray machines with uGNI: XE, XK, XC
- Sun et al., A uGNI-Based Asynchronous Message-driven Runtime System for Cray Supercomputers with Gemini Interconnect, IPDPS 2012
- Sun et al., 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 et al., Acceleration of an Asynchronous Message Driven Programming Paradigm on IBM Blue Gene/Q, IPDPS 2013

Machines supporting MPI

Infiniband clusters
Performance - Bandwidth on BGQ

The diagram shows the bandwidth (in GBytes/sec) for different file sizes (1024 Bytes, 32K Bytes, 1M Bytes) across four Charm++ architectures: PAMI, PAMI-LRTS, PAMI SMP, and PAMI-LRTS SMP. The bandwidth remains relatively constant across all architectures and file sizes.
Application Performance

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

<table>
<thead>
<tr>
<th>Charm++ architectures</th>
<th>Timestep (ms/step)</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 Nodes (2048 hw threads)</td>
<td>(10)</td>
</tr>
<tr>
<td>64 Nodes (4096 hw threads)</td>
<td>(9)</td>
</tr>
</tbody>
</table>

Yanhua Sun  U of Illinois at Urbana-Champaign  22/24
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

- 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
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