Optimizing non-blocking Collective Operations for InfiniBand

Torsten Hoefler and Andrew Lumsdaine

Open Systems Lab
Indiana University
Bloomington, USA

IPDPS'08 - CAC'08 Workshop

Miami, FL, USA

April, 14th 2008

Introduction

Non-blocking collective operations (NBC) are beneficial to:

- hide communication latency by overlapping
- use the available bandwidth better
- avoid detrimental effects of pseudo-synchronization/process skew
- make efficient use of the new semantics

LibNBC and MPI

LibNBC implements all MPI collective operations in a non-blocking way on top of non-blocking MPI point-to-point (p2p) functions.

Introduction

Non-blocking collective operations (NBC) are beneficial to:

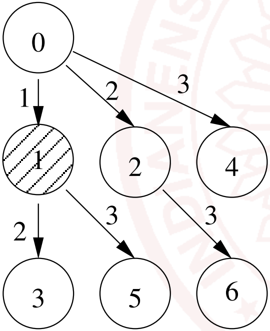
- hide communication latency by overlapping
- use the available bandwidth better
- avoid detrimental effects of pseudo-synchronization/process skew
- make efficient use of the new semantics

LibNBC and MPI

LibNBC implements all MPI collective operations in a non-blocking way on top of non-blocking MPI point-to-point (p2p) functions.

Schedule-based design:

- a process-local schedule of p2p operations is created for every collective operation
- example; 7-process bcast, schedule on rank 1:



Pseudocode for schedule at rank 1:

```
NBC_Sched_recv(buf, count, dt, 0, schedule);
NBC_Sched_barr(schedule);
NBC_Sched_send(buf, count, dt, 3, schedule);
NBC_Sched_barr(schedule);
NBC_Sched_send(buf, count, dt, 5, schedule);
```

schedule in memory:

recv from 0	end	send to 3	end	send to 5

Progress or no Progress?

Progress is most important for efficient overlap! LibNBC has two levels:

LibNBC Progress

- schedule execution is represented as a state machine
- state and schedule are attached to every request
- schedules might be cached/reused
- progression in NBC_Test, NBC_Wait

MPI Progress

- progress the MPI communication protocol
- (a)synchronous progress?
- progress has to be made in every MPI call
- LibNBC scheduler calls MPI_Testall in NBC_Test/NBC_Wait

MPI Progress?

- focus on transport-layer (MPI) progress
- many MPI implementations don't support asynchronous progress well
- some do (MVAPICH, Open MPI) but MPI peculiarities cause high overhead
- LibNBC only requires a small subset of MPI
- $\Rightarrow$ define and implement mini-MPI

MPI has problems? No ...

- MPI_ANY_SOURCE enforces sender-based rendezvous protocol (three messages instead of two in the receiver based case)
- $\Rightarrow$ MPI-3 subsetting might help (later)!

LibNBC's needs?

- non-blocking send (starts a send operation with low CPU overhead)
- non-blocking receive (post a receive or receive data with low CPU overhead, sender is known)
- request objects to identify the outstanding operations
- communication contexts (similar to MPI communicators)
- message tags (tags are needed to identify operation)
- message ordering must be guaranteed
- test for completion (very low overhead!)
- wait for completion (might `sched_yield()`)

Specialized InfiniBand^{TM} Transport Layer

- mini-MPI for InfiniBand
- InfiniBand's message transmission is fully asynchronous (once the Work Request (WR) is posted)
- posting a WR is cheap ($\approx$ 100ns)
- uses RDMA-W (known scalability issues)
- eager and rendezvous protocol
- $\rightarrow$ eager protocol is fully asynchronous (if credits are available on receiver)
- $\rightarrow$ rendezvous protocol is more complex (next slide)

The Rendezvous Protocol

Minimize the number of synchronization points:

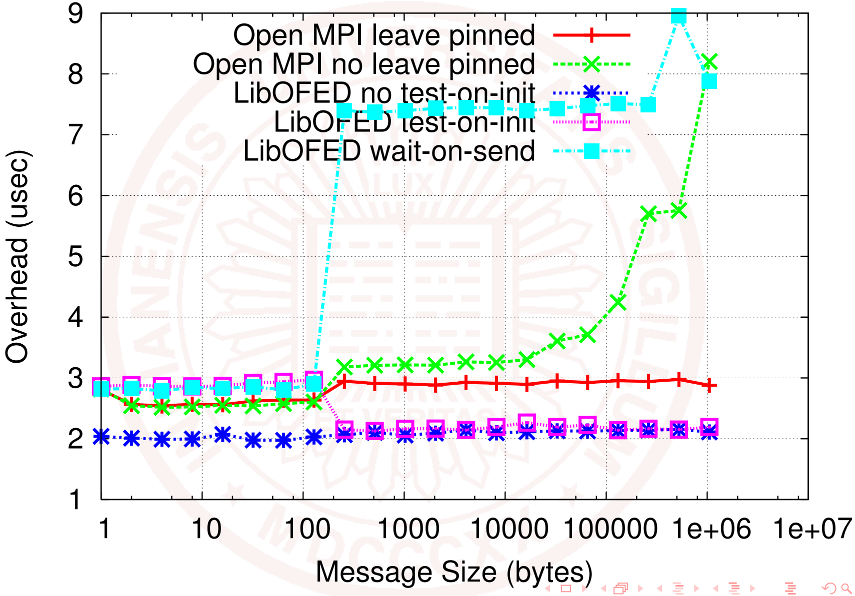
- receiver-driven protocol (LibOF):
  1. receiver sends RTR to sender (`addr,r_key`)
  2. sender sends data after receiving RTR
  3. one synchronization point

... problematic if sender arrives after receiver!

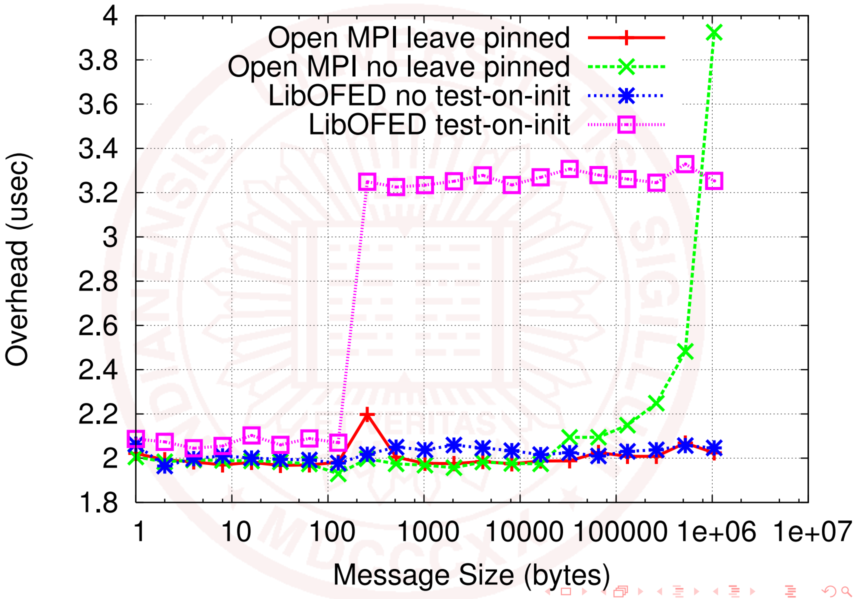
Two Progression Optimization Strategies

- test-on-init (polls all CQ at the end of `OF_Isend()` and `OF_Irecv()`)
- wait-on-send (polls a defined time in `OF_Isend()`)

Netgauge overhead benchmarks - `OF_Isend()`



Netgauge overhead benchmarks - `OF_Irecv()`



Optimizing Wait-on-send for LibNBC

- wait-on-send adds up to 5 μs per message to the CPU overhead
- LibNBC often issues multiple messages
- problematic for many messages (huge communicators)
- implemented `OF_Startall` which starts multiple messages (like wait-on-send for multiple messages)
- is called after all messages are posted
- times out (to avoid deadlocks)

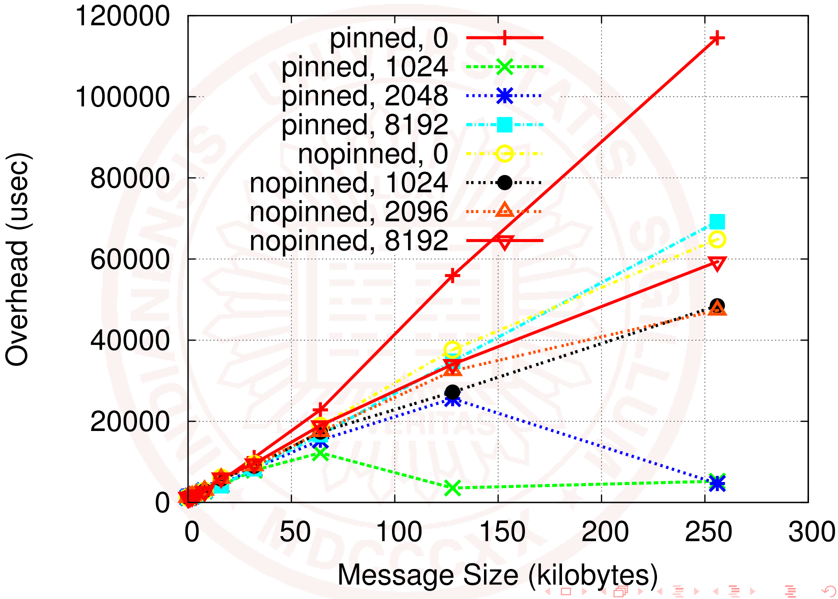
Progression Strategies

- MPI and unoptimized LibOF library must be called to make progress
- libraries might use pipelined transfers (Open MPI does)
- $\rightarrow$ test frequency depends on message size
- number of tests $N = \left\lfloor \frac{size}{interval} \right\rfloor + 1$
- we tested all size-intervals between 0 (no tests) and 32*kiB*

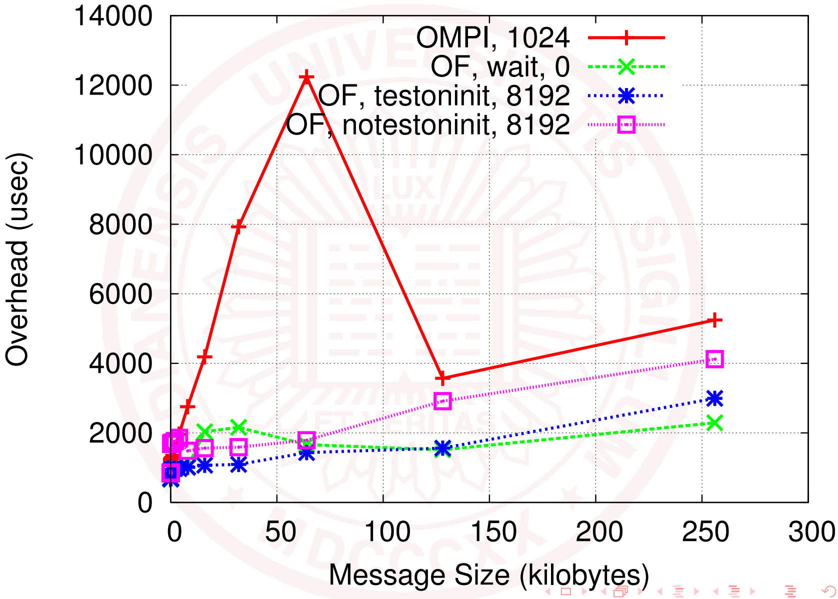
Benchmarks with NBCBench

1. NBCBench takes the latency of a blocking operation ϵ
2. issue a non-blocking operation
3. compute for time ϵ (and issue *N* equi-distant tests)
4. wait for operation to finish
5. report times for step $2 + 4$ as overhead

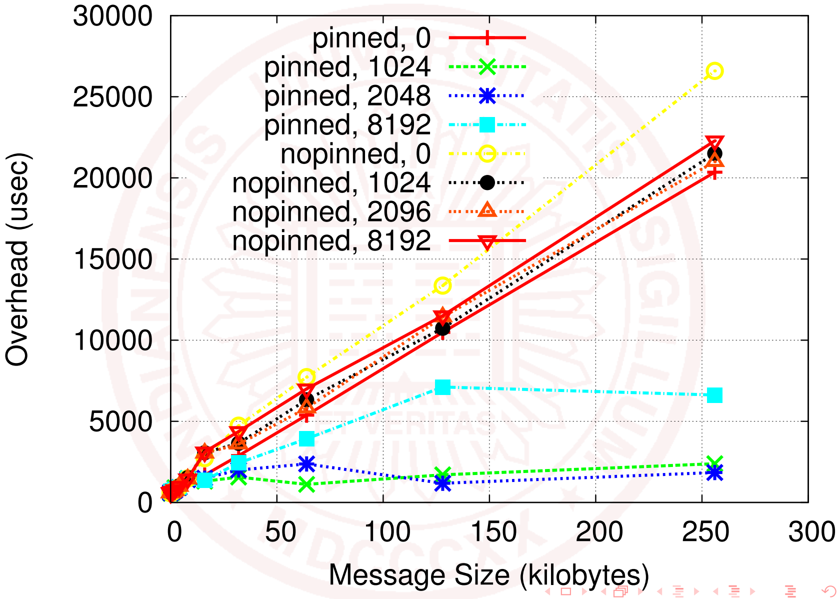
NBCBench with Open MPI - NBC_Ialltoall on 64 nodes



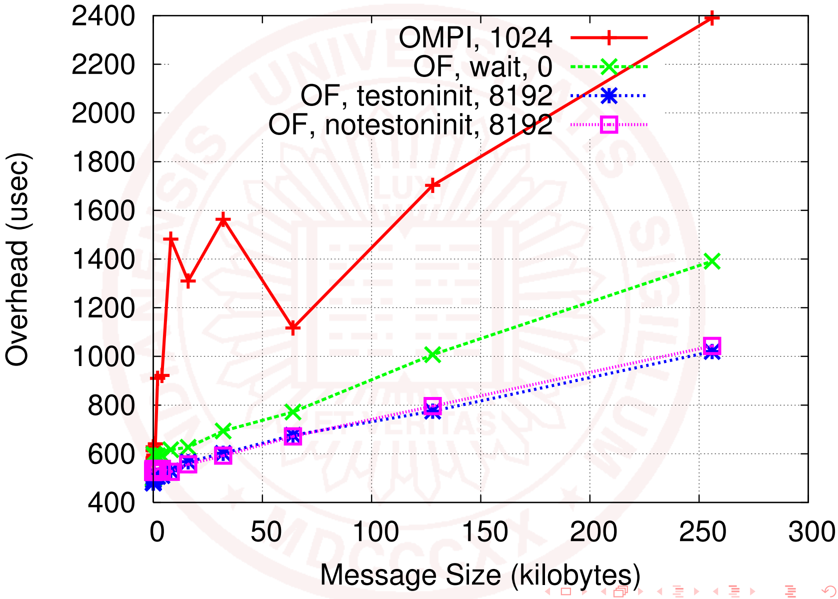
NBCBench with LibOF - NBC_Ialltoall on 64 nodes



NBCBench with Open MPI - NBC_Igather on 64 nodes



NBCBench with LibOF - NBC_Igather on 64 nodes



Application Kernel Results

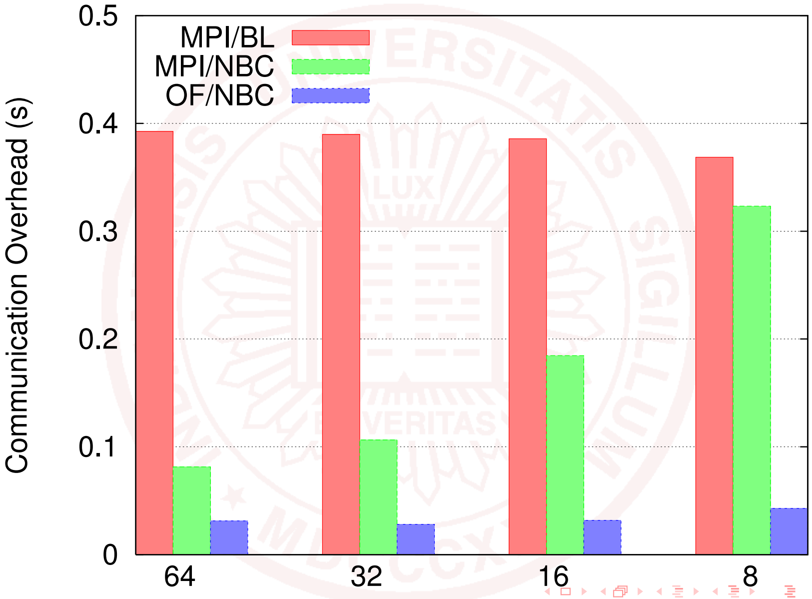
Parallel Compression

1. compress data in parallel
2. gather it to a single host in a pipelined fashion vs. single gather in MPI case
3. overlap with NBC_Igather

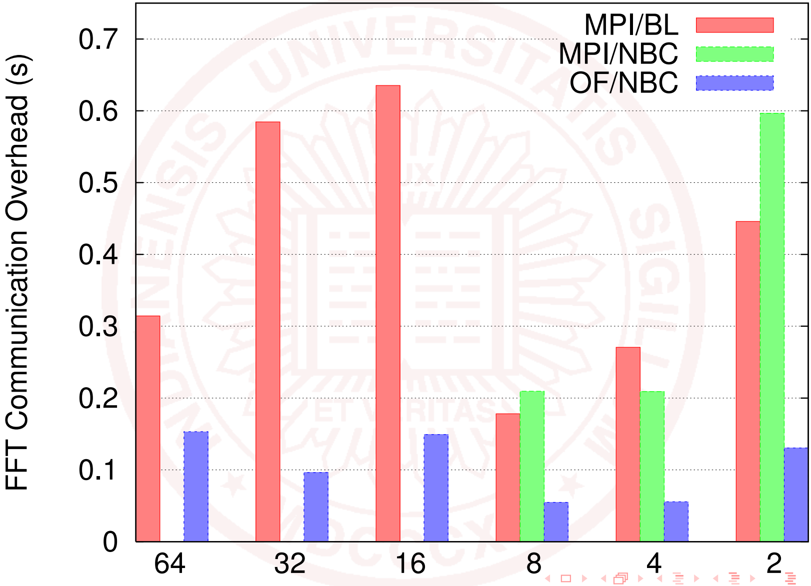
Three-dimeonsional FFT

1. transform in two dimensions, transpose with MPI_Alltoall and transform third dimension in MPI case
2. transform plane-by-plane and pipeline communication with NBC_Ialltoall (overlap)

Parallel Compression Communication Overhead



Parallel tree-dimensional FFT



Conclusions and Future Work

Conclusions

- defined LibNBC's requirements for transport interface
- implemented overlap-optimized InfiniBand transport
- proposed and evaluated different optimizations to enhance asynchronous progression
- showed significant performance improvements in microbenchmarks as well as application kernels

Future Work

- implement high-overlap support for different networks
- evaluate threaded progression strategies
- offload scheduler operations/state machine to the network

Conclusions and Future Work

Conclusions

- defined LibNBC's requirements for transport interface
- implemented overlap-optimized InfiniBand transport
- proposed and evaluated different optimizations to enhance asynchronous progression
- showed significant performance improvements in microbenchmarks as well as application kernels

Future Work

- implement high-overlap support for different networks
- evaluate threaded progression strategies
- offload scheduler operations/state machine to the network