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
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.
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
- ⇒ 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)
- ⇒ 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
- testall for completion (very low overhead!)
- waitany for completion (might `sched_yield()`)

Torsten Hoefler and Andrew Lumsdaine
LibNBC for InfiniBand
Specialized InfiniBand™ 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 (≈ 100ns)
- uses RDMA-W (known scalability issues)
- eager and rendezvous protocol
  - eager protocol is fully asynchronous (if credits are available on receiver)
  - 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 \(\text{OF\_Isend()}\) and \(\text{OF\_Irecv()}\))
- wait-on-send (polls a defined time in \(\text{OF\_Isend()}\))
Netgauge overhead benchmarks - \texttt{OF\_Isend()}
wait-on-send adds up to 5 $\mu s$ per message to the CPU overhead
LibNBC often issues multiple messages
problematic for many messages (huge communicators)
implemented \texttt{OF\_Start\_all} 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)
  → test frequency depends on message size
- number of tests \( N = \left\lfloor \frac{\text{size}}{\text{interval}} \right\rfloor + 1 \)
- we tested all size-intervals between 0 (no tests) and 32\(kiB\)

Benchmarks with NBCBench

1. NBCBench takes the latency of a blocking operation \( \varepsilon \)
2. issue a non-blocking operation
3. compute for time \( \varepsilon \) (and issue \( N \) equi-distant tests)
4. wait for operation to finish
5. report times for step 2 + 4+ testtime as overhead
NBCBench with Open MPI - NBC_Ialltoall on 64 nodes

Overhead (usec) vs. Message Size (kilobytes)

- pinned, 0
- pinned, 1024
- pinned, 2048
- pinned, 8192
- nopinned, 0
- nopinned, 1024
- nopinned, 2096
- nopinned, 8192
NBCBench with LibOF - NBC_Ialltoall on 64 nodes

- Overhead (usec) vs. Message Size (kilobytes)
- Data points and lines for different configurations:
  - OMPI, 1024
  - OF, wait, 0
  - OF, testoninit, 8192
  - OF, notestoninit, 8192

Torsten Hoefler and Andrew Lumsdaine
LibNBC for InfiniBand
NBCBench with LibOF - NBC_Igather on 64 nodes

Message Size (kilobytes)

Overhead (usec)

OMPI, 1024
OF, wait, 0
OF, testoninit, 8192
OF, notestoninit, 8192

Torsten Hoefler and Andrew Lumsdaine
LibNBC for InfiniBand
**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-dimensional 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

![Graph showing communication overhead for different MPI/NBC and OF/NBC configurations with varying number of processes.]

- MPI/BL
- MPI/NBC
- OF/NBC

Communication Overhead (s)

- Number of processes: 64, 32, 16, 8
Parallel tree-dimensional FFT

FFT Communication Overhead (s)

<table>
<thead>
<tr>
<th></th>
<th>MPI/BL</th>
<th>MPI/NBC</th>
<th>OF/NBC</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>0.3</td>
<td>0.2</td>
<td>0.1</td>
</tr>
<tr>
<td>32</td>
<td>0.6</td>
<td>0.5</td>
<td>0.4</td>
</tr>
<tr>
<td>16</td>
<td>0.7</td>
<td>0.6</td>
<td>0.5</td>
</tr>
<tr>
<td>8</td>
<td>0.4</td>
<td>0.3</td>
<td>0.2</td>
</tr>
<tr>
<td>4</td>
<td>0.2</td>
<td>0.1</td>
<td>0.05</td>
</tr>
<tr>
<td>2</td>
<td>0.1</td>
<td>0.08</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Torsten Hoefler and Andrew Lumsdaine

LibNBC for InfiniBand
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

- 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