Motivation

- Previous work on DD-$\alpha$AMG on QPACE 2/Xeon Phi has shown that after code optimizations, off-chip communication became dominant [1]
- This applies not only to DD-$\alpha$AMG but also to other Lattice QCD applications and beyond
- Encountered issues with various MPI implementations
  - Context switches due to non-pinnable MPI internal threads
  - Missing support for non-standard network topologies

Objectives

- Use persistent communication
- Use one-sided communication (RDMA hardware capabilities)
- Reduce software-induced latency to a bare minimum
- No extra thread pool
- Fast adaptation to new hardware
- Support for exotic network topologies
- Vendor independent
- Optimize for typical lattice QCD communication patterns
- De facto drop-in replacement for MPI

Implementation

Features

- Buffered and unbuffered point-to-point data transfers
- Global reduction with user-defined functions (e.g., global sum)
- Auxiliary functions for communicator setup and usage
- Modern C++11
- Avoid code dependencies as far as possible
- Separate code for each supported network provider (IB verbs, Linux CMA)
- Can easily add or remove providers
- Limited C interface for compatibility with existing software
- Allow for compile-time optimization
- Set provider and topology settings at compile time
  - Each binary is cluster specific
- No polymorphism (avoid viable lookup)

Code comparison – MPI (left) vs. pMR (right)

```c
// Setup persistent communication channel
pMR::Connection connection(pMR::Target(...));

// Send request
MPI_Request sendRequest;
pMR::SendWindow<float>(sendRequest, connection, sendBuffer, count);

// Receive request
MPI_Request recvRequest;
pMR::RecvWindow<float>(recvRequest, connection, recvBuffer, count);

for(i = start; i != end; ++i)
{
  // Computation
  recvWindow.init();
  sendWindow.init();

  // Computation
  MPI_Irecv(recvRequest, ...);
  sendWindow.post();

  // Computation
  MPI_Wait(sendRequest, ...);
  recvWindow.wait();

  // Computation
}
```

Real-world benchmark: DD-$\alpha$AMG [2]

- Choose a test case that is communication bound: DD-$\alpha$AMG coarse-grid solve
- In current implementation, coarse grid is spread over entire machine
- In future implementation, coarse grid could be mapped onto subset of machine, but coarse-grid solve would still be communication bound

Modifications to halo exchange code

- Use buffered send and receive for all exchanges
- Allows for the re-use of buffers
  - Persistent communication possible without major code changes
  - Downside: doubles the overhead for copies from/to buffers
- Replace MPI calls with corresponding pMR calls
  - Only minor code changes

Results: Benchmark details

- CLS lattice: $8^4 \times 96$, $\beta = 3.4$, $m_s = 220$ MeV, $\alpha = 0.086$ fm [3]
- Small lattice chosen intentionally to see breakdown of strong scaling
- Four Intel Xeon Phis per node
- Infiniband FDR 1D Flexible Hyperblock Torus Topology
- Uses Intel Xeon Phi for computation exclusively (native programming model)

Results: Halo exchange on coarse grid for one solve

![Halo exchange graph](image)

Results: Coarse grid contributions – MPI (left) vs. pMR (right)

![Coarse grid contributions graph](image)

Conclusion

- pMR can be used in existing software with only minor code changes
- Reduces impact of communication without algorithmic changes

Future opportunities

- Use pMR for global sums in DD-$\alpha$AMG
- Use pMR in other communication-bound applications

Interested?

- Checkout repository [NOW](https://rqcd.ur.de:8443/gep21271/pmr)
- Licensed under Apache License 2.0
- Will be opened up for contributions from anybody (GitHub)

References