pMR: A high-performance communication library

Peter Georg, Daniel Richtmann, and Tilo Wettig
Department of Physics, University of Regensburg

Motivation

- Previous work on DD-αAMG on QPACE 2/Xeon Phi has shown that after code optimizations, off-chip communication became dominant [1]
- This applies not only to DD-αAMG 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
    $\Rightarrow$ Each binary is cluster specific
  - No polymorphism (avoid vtable lookup)

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

```
MPI_Request sendRequest;

MPI_Request recvRequest;


for(i = start; i != end; ++i)
{




    // Computation

    MPI_Irecv(recvBuffer, ...);
    MPI_Isend(sendBuffer, ...);

    // Computation

    MPI_Wait(sendRequest, ...);
    MPI_Wait(recvRequest, ...);

    // Computation
}
```

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

pMR::SendWindow<float>
    sendWindow(connection, sendBuffer, count);
pMR::RecvWindow<float>
    recvWindow(connection, recvBuffer, count);

for(i = start; i != end; ++i)
{
    // Computation

    recvWindow.init();
    sendWindow.init();

    // Computation

    sendWindow.post();
    recvWindow.post();

    // Computation

    sendWindow.wait();
    recvWindow.wait();

    // Computation
}
```

References

[1] S. Heybrock et al., *Adaptive algebraic multigrid on SIMD architectures*, PoS **LATTICE2015** (2016) [arXiv:1512.04506].

[2] A. Frommer et al., *Adaptive Aggregation Based Domain Decomposition Multigrid for the Lattice Wilson Dirac Operator*, SIAM J. Sci. Comput. **36** (2014) A1581 [arXiv:1303.1377].

[3] M. Bruno et al., *Simulation of QCD with $N_f = 2 + 1$ flavors of non-perturbatively improved Wilson fermions*, JHEP **02** (2015) 043 [arXiv:1411.3982].

[4] P. Arts et al., *QPACE 2 and Domain Decomposition on the Intel Xeon Phi*, PoS **LATTICE2014** (2015) 021 [arXiv:1502.04025].

Real-world benchmark: DD-αAMG [2]

- Choose a test case that is communication bound: DD-αAMG 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

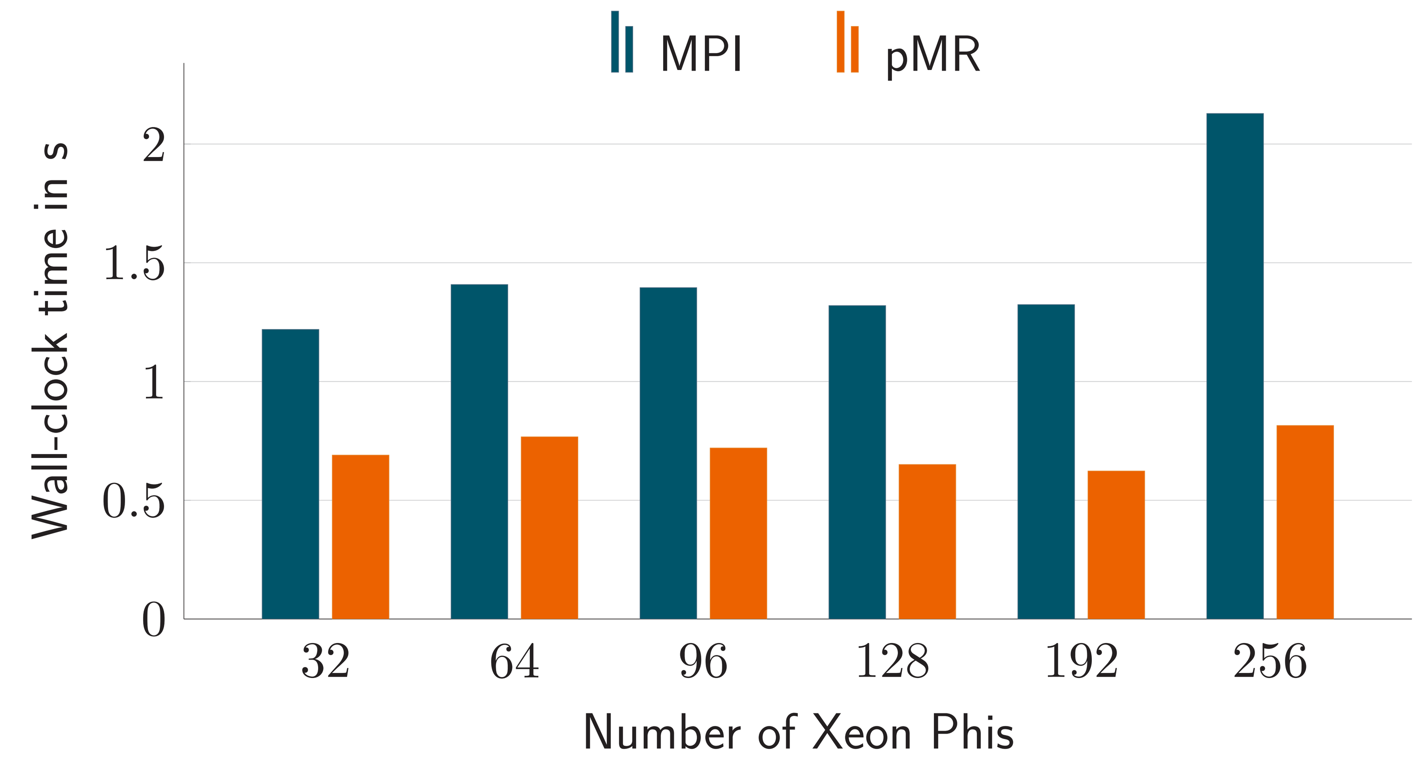
Modifications to halo exchange code

- Use buffered send and receive for all exchanges
  - Allows for the re-use of buffers
    $\Rightarrow$ 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

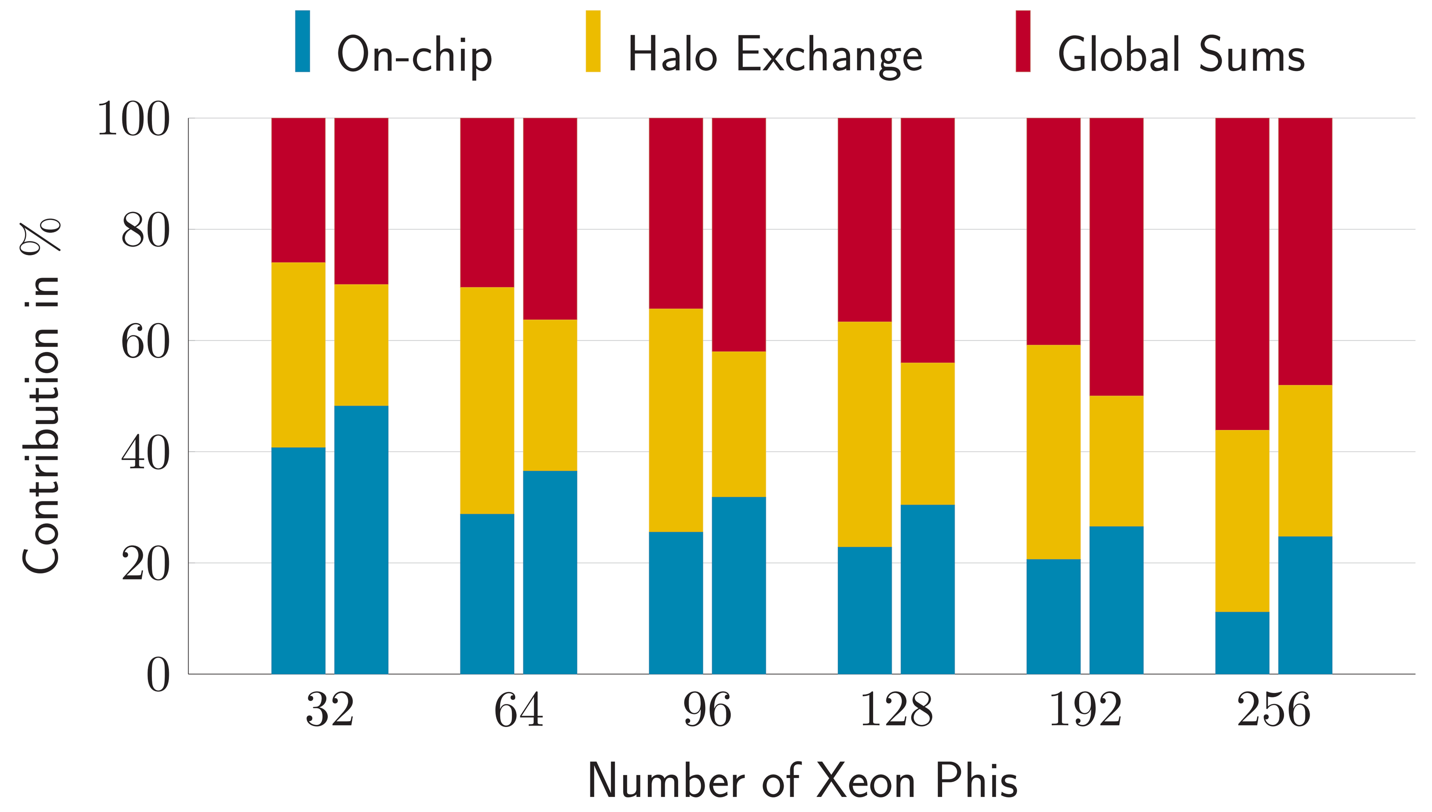
Results: Benchmark details

- CLS lattice: $48^3 \times 96$, $\beta = 3.4$, $m_\pi = 220\,\mathrm{MeV}$, $a = 0.086\,\mathrm{fm}$ [3]
  - Small lattice chosen intentionally to see breakdown of strong scaling
- QPACE 2: Intel Xeon Phi cluster [4]
  - Four Intel Xeon Phis per node
  - Infiniband FDR 1D Flexible Hyperblock Torus Topology
  - Uses Intel Xeon Phis for computation exclusively (native programming model)

Results: Halo exchange on coarse grid for one solve



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



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-αAMG
- 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)

Supported by the German Research Foundation (DFG) in the framework of SFB/TR-55
peter.georg@ur.de