View-aware Message Passing Through the Integration of Kokkos and ExaMPI

Evan Drake Suggs (Speaker)
Jan Ciesko
Stephen L. Olivier
Anthony Skjellum

Introduction

- Advanced C++ libraries/code are rapidly being adopted in HPC, but many are constrained by their use with C libraries not designed for interacting with them.
- Message Passing Interface (MPI) standard specifies a programming model for passing messages between processes.
- ExaMPI is a modern C++ focused implementation of the MPI Standard.
- Kokkos is a programming model for C++ that provides data structures, concurrency features, and algorithms to support advanced C++ parallel programming across different memory spaces.
- MPI is an inter-process/node parallel programming model while Kokkos is an intra-process/node parallel programming model.
- However, parallel applications want to use both.

Goals

- To improve the general programming experience when using MPI with Kokkos.
- To minimize the possibility of bugs from MPI+Kokkos programs
- To enable optimizations for MPI+Kokkos at the language binding level or below.

ExaMPI is focused on principles-first design, highlighting the principles below:

- • Enable rapid new development of new features, identify ways to increase performance, and improve understanding of the MPI standard
- • Support the research interests and experiments of developers, such as effective overlap of communication and computation

Goals cont.

- Create an MPI Extension composed of a series of function bindings within ExaMPI to handle a Kokkos View as input or output in a similar way to buffers in MPI.
- Interface should not require the user to touch the .data() method.
- Ensure the MPI+Kokkos extension with comparable performance to traditional methods
- Due to additional handling cost, performance may be slightly worse, but has the benefit of functionality.
- Gather ideas for what further work can be done, and how feasible it is to create a larger extension

Objectives

Our objectives are as follows:

- To create a series of function bindings within ExaMPI whose syntax utilizes Kokkos objects in the same manner as standard MPI buffers
- These function bindings should have at least comparable performance to existing practices for the majority of use cases
- Allow for easier building of MPI applications using Kokkos alongside ExaMPI

Objectives cont.

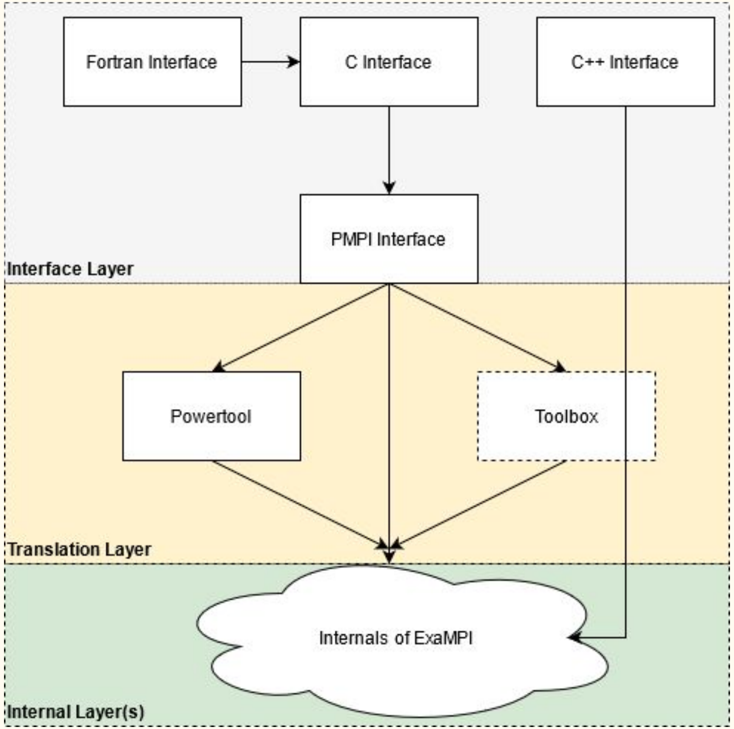
These objectives are accompanied by the following questions:

- How useful are these new bindings for users?
- What are the long term opportunities created by these bindings?
- Should these bindings follow the more traditional C-style MPI bindings or experiment with new parameters?
- Do these bindings increase or decrease performance?

Literature Review

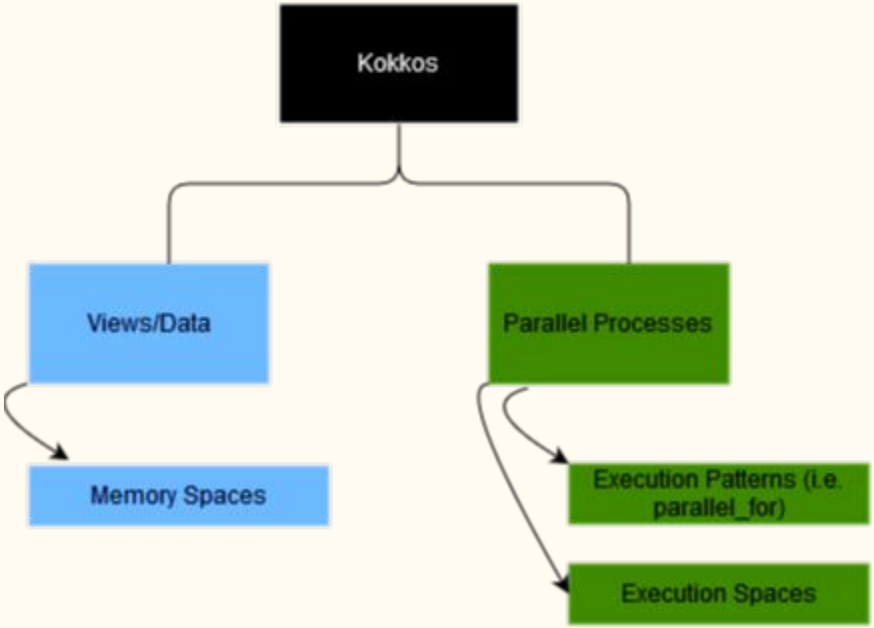
MPI and ExaMPI

- The Message Passing Interface (MPI) standard specifies a programming model interface for passing messages between peer processes.
- MPI offers a wide range of functions, but the most common are the point-to-point functions, MPI_Send and MPI_Recv.
- Buffers in MPI are often a contiguous array of a single type of data primitive.
- ExaMPI is a modern C++ implementation of MPI, which enabled this paper to put C++ features, such as templates, into the work.
- ExaMPI is focused on being a tool for rapid research, implementing a subset of MPI functions rather than the full standard.



Kokkos

- Kokkos is a programming model and library that provides data structures, concurrency features and algorithms to support advanced C++ parallel programming across different memory spaces.
- Kokkos is centered around its primary data structure, the View, a smart pointer class wrapping primitive datatypes arrays
- Kokkos has a few parallel dispatch operations similar to those used in OpenMP: `parallel_for`, `parallel_reduce`, and `parallel_scan`.



Kokkos

- Coding example below shows initialization of a Kokkos View.

```
Kokkos::View<double*>check( ”check”, n );
Kokkos::parallel_for(check.extent(0), KOKKOS LAMBDA(int i) {
   check(i) = i*i;
});
```

Related Work

- There exist a few previous attempts unite MPI or MPI alternatives with Kokkos
- Khuvis et al. have a shown a speedup of General Matrix Multiplication (GEMM) code and the Graph500 benchmark using MPI+Kokkos.
- The GEMM code uses Kokkos for parallelism of matrix multiplication alongside MPI to distribute the matrices, this code has noticeable improvement with each additional process for up to 64 processes.
- The Graph500 results show a speed-up with the locking implementation over the MPI-only one up to forty processes, and continued speed-up with the non-locking implementation of up to 5x on 64 processes.

Related Work

- The Uintah framework for modeling chemical reactions uses MPI + Kokkos to consolidate the use of both MPI + Pthreads and MPI + CUDA into a single approach that also enables added portability.
- The UPC++ framework replaces MPI in simulating heat conduction without radical changes in performance compared to the IBM version of MPI.
- Con et al. demonstrated use of the distributed many-task MPI alternative Legion with Kokkos to offload "boiler-plate code" away from the user.
- The primary innovation that distinguishes this project from previous forms of MPI+Kokkos interaction is the ability to take Kokkos Views directly.

Methodology and Implementation

Further Requirements

- Able to handle Views with up to 3 dimensions
- Able to handle any contiguous View regardless of layout
- The interface should not require the user to create new derived datatypes for Views
- Retain datatype compatibility with Kokkos
- Avoid reliance on the MPI Datatype by using template parameters

MPI_Kokkos_Send & Recv Implementation

MPI_Kokkos_Send⟨View_t, Datatype⟩(

View_t * buf, int count, MPI Datatype datatype, int dest, int tag, MPI Comm comm)

MPI_Kokkos_Recv⟨View_t, Datatype⟩(

View_t * buf, int count, MPI Datatype datatype, int source, int tag, MPI Comm comm)

- The template parameters decide the datatype information for View operations
- MPI_Kokkos_Send's counterpart, MPI_Kokkos_Recv receives the Payload send in MPI_Kokkos_Send, then wraps that in a View object and sends that to the pointer passed as a parameter.

```
Kokkos::View<int*>check( ”check”, n );
MPI_Kokkos_Send<Kokkos::View<int*>, int>(&check, n, MPI_INT, 0, 0, MPI_COMM_WORLD);
int* check_arr = check.data();
MPI_Send(check_arr, n, MPI_INT, 0, 0, MPI_COMM_WORLD);
```

Further Implementations

- MPI_Kokkos_ISend⟨View_t, Datatype⟩(View_t * buf, int count, MPI Datatype datatype, int dest, int tag, MPI Comm comm, MPI Request *request)
- MPI_Kokkos_Irecv⟨View_t, Datatype⟩(View_t * buf, int count, MPI Datatype datatype, int dest, int tag, MPI Comm comm, MPI Request *request)
  - Synchronizes with standard functions (i.e. MPI_Wait)
- MPI_Kokkos_Bcast⟨View_t, Datatype⟩(View_t * buf, int count, MPI Datatype datatype, int root, MPI Comm comm)
- MPI_Kokkos_Allgather⟨View_t, Datatype⟩(View_t * buf, int count, MPI Datatype datatype, View_t * recv buf, int recv_count, MPI_Datatype recv_type, MPI_Comm comm, MPI_Request *request)
  - MPI Kokkos Allgather takes (gathers) an input View from all processes.
  - Then, these are compiled into a View ordered by their sending process's index ranking.
- MPI_Kokkos_Allreduce⟨View_t, Datatype⟩(View_t * buf, View_t * recv_buf, int count, MPI Datatype datatype, MPI_Op op, MPI_Comm comm)
  - The key design choice is whether this extension should return the resulting buffer from the reduce operation as a View or as a data primitive.
  - To be more consistent with the previously covered functions, this function uses Views for both the send and receive buffers

MPI_Kokkos_Bcast Implementation

MPI_Kokkos_Bcast⟨View_t, Datatype⟩(View_t * buf, int count, MPI Datatype datatype, int root, MPI Comm comm)

- MPI_Kokkos_Bcast (short for Broadcast) resembles the functionality of Send and Receive combined into one function.
- The root process sends (Broadcasts) its View to all the other process ranks in the given communicator group, enabling a process to send to any number of other processes.
- Broadcast functions are a form of collective communication as it handles several processes.
- In communicator groups, only one process rank is identified as the root process.

MPI_Kokkos_Allgather Implementation

MPI_Kokkos_Allgather⟨View_t, Datatype⟩(View_t * buf, int count, MPI Datatype datatype, View_t * recv buf, int recv_count, MPI_Datatype recv_type, MPI_Comm comm, MPI_Request *request)

- MPI Kokkos Allgather takes (gathers) an input View from all processes.
- Then, these are compiled into a View ordered by their sending process's index ranking.
- MPI_Allgather functions are classified as a form of collective communication as it handles several processes.

MPI_Kokkos_Allreduce Implementation

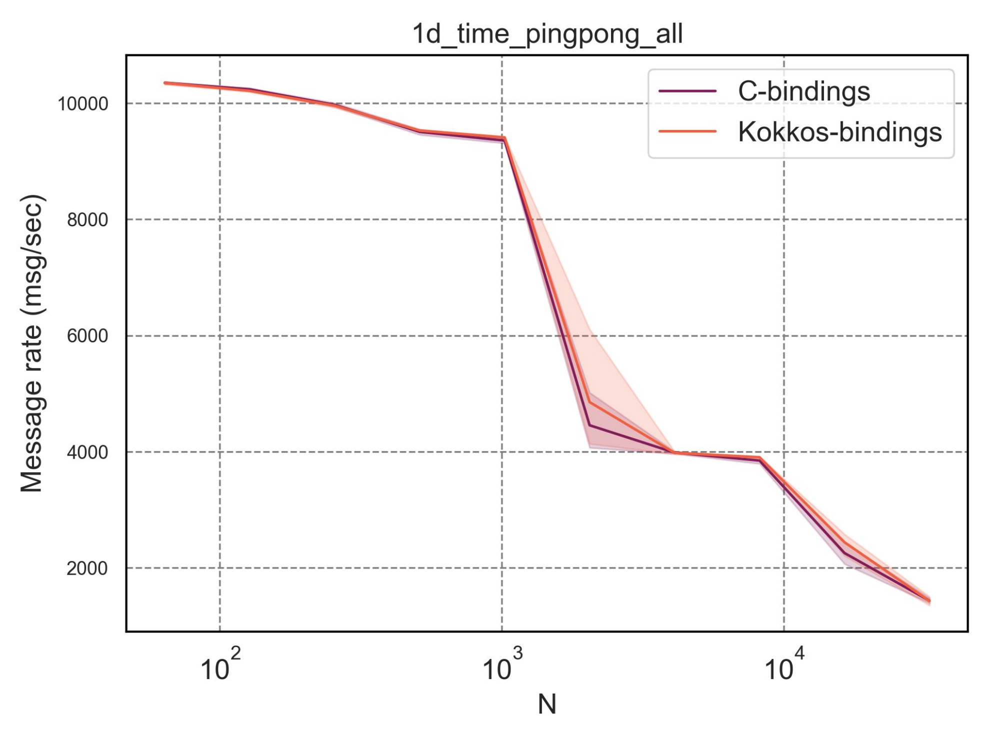
MPI_Kokkos_Allreduce⟨View_t, Datatype⟩(View_t * buf, View_t * recv_buf, int count, MPI Datatype datatype, MPI_Op op, MPI_Comm comm)

- MPI_Kokkos_Allreduce is a collective function that collects values from several processes, performs an operation on them (MPI_Op) and broadcasts the result to all processes involved.
- The MPI_Op can be any of a number of operations such as sum, max, etc. This function required more conceptual work than previous function.
- The key design choice is whether this extension should return the resulting buffer from the reduce operation as a View or as a data primitive.
- To be more consistent with the previously covered functions, this function uses Views for both the send and receive buffers

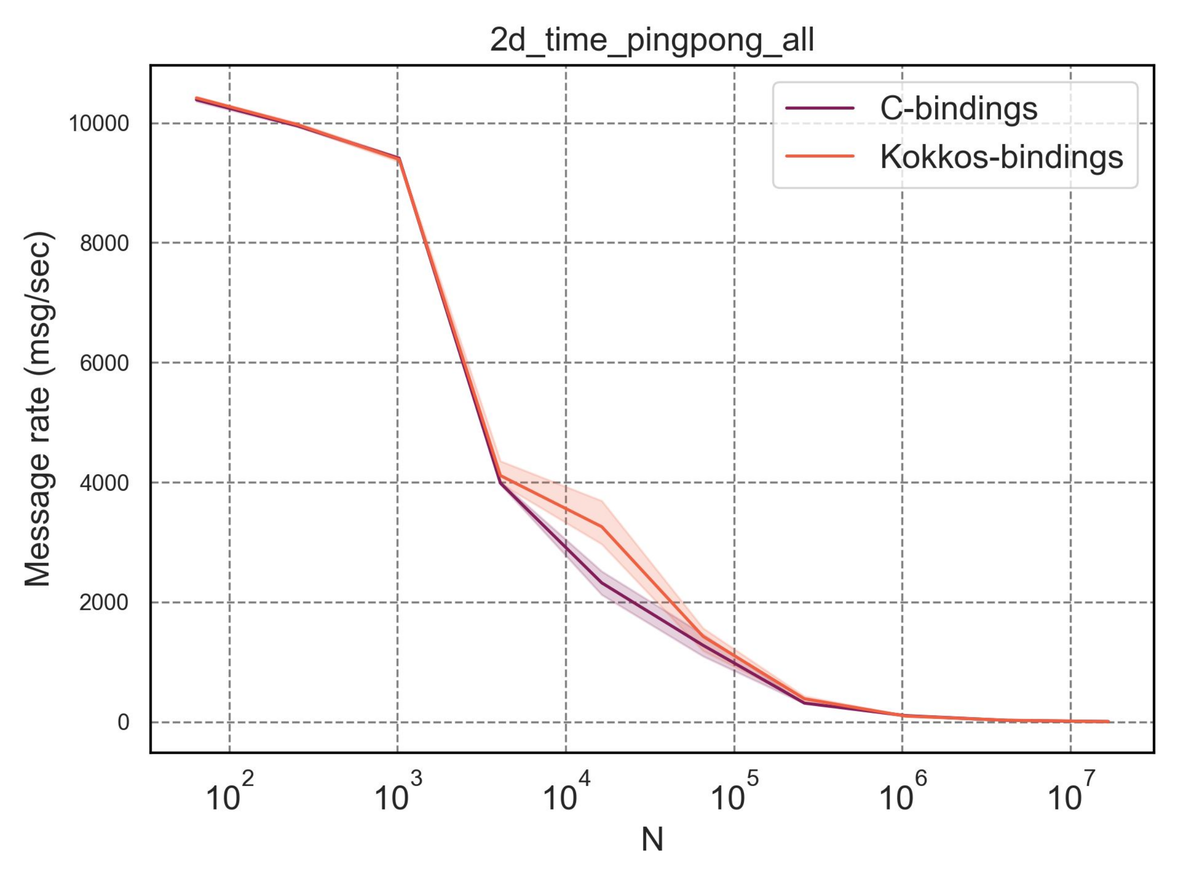
Test Results

Single-Dimension Ping-Pong Tests

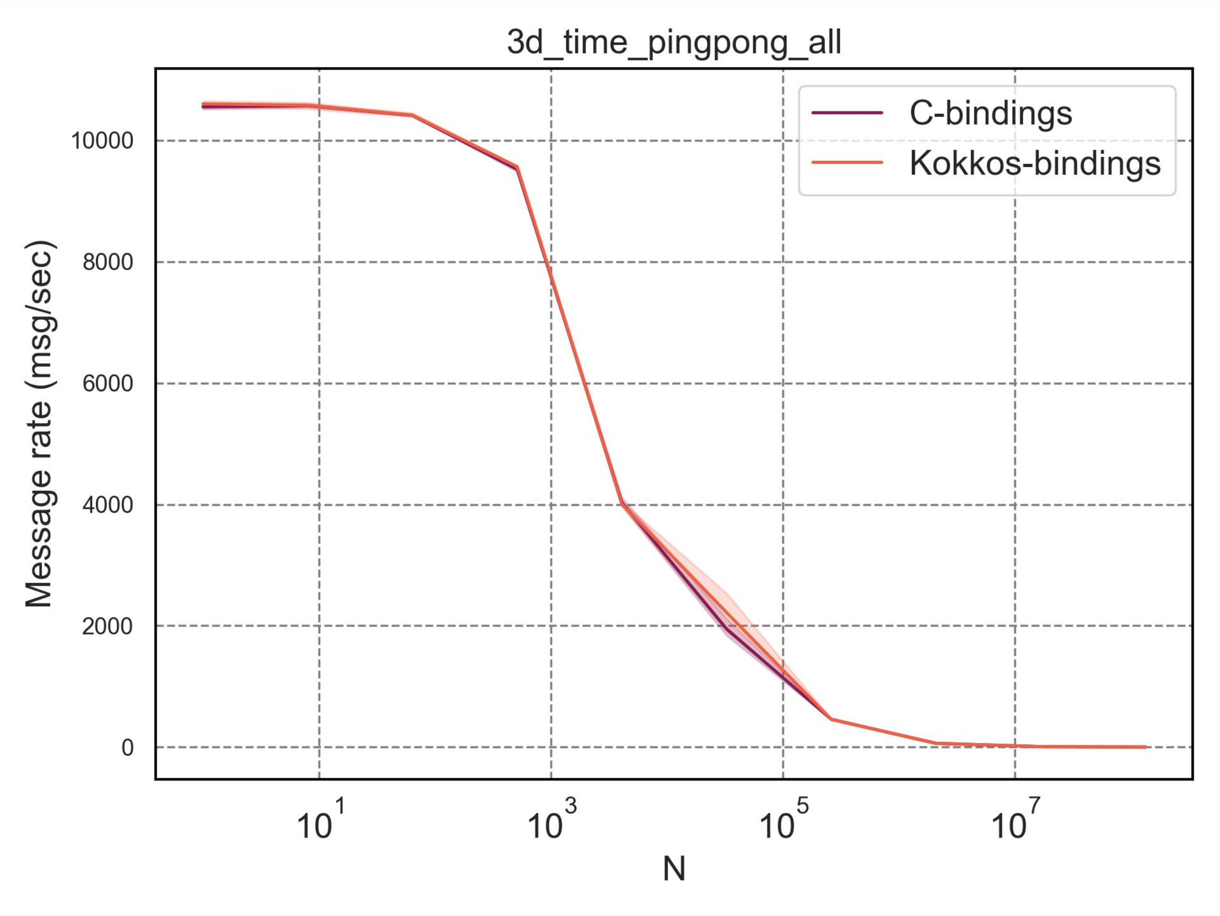
- 500 runs were averaged on Views with from 64 to 32768 elements
- The primary goal was not a significant change in performance, but roughly equal performance for both bindings
- Both types of bindings are better at different times, with very low standard deviation of the mean



Two-Dimensional Tests

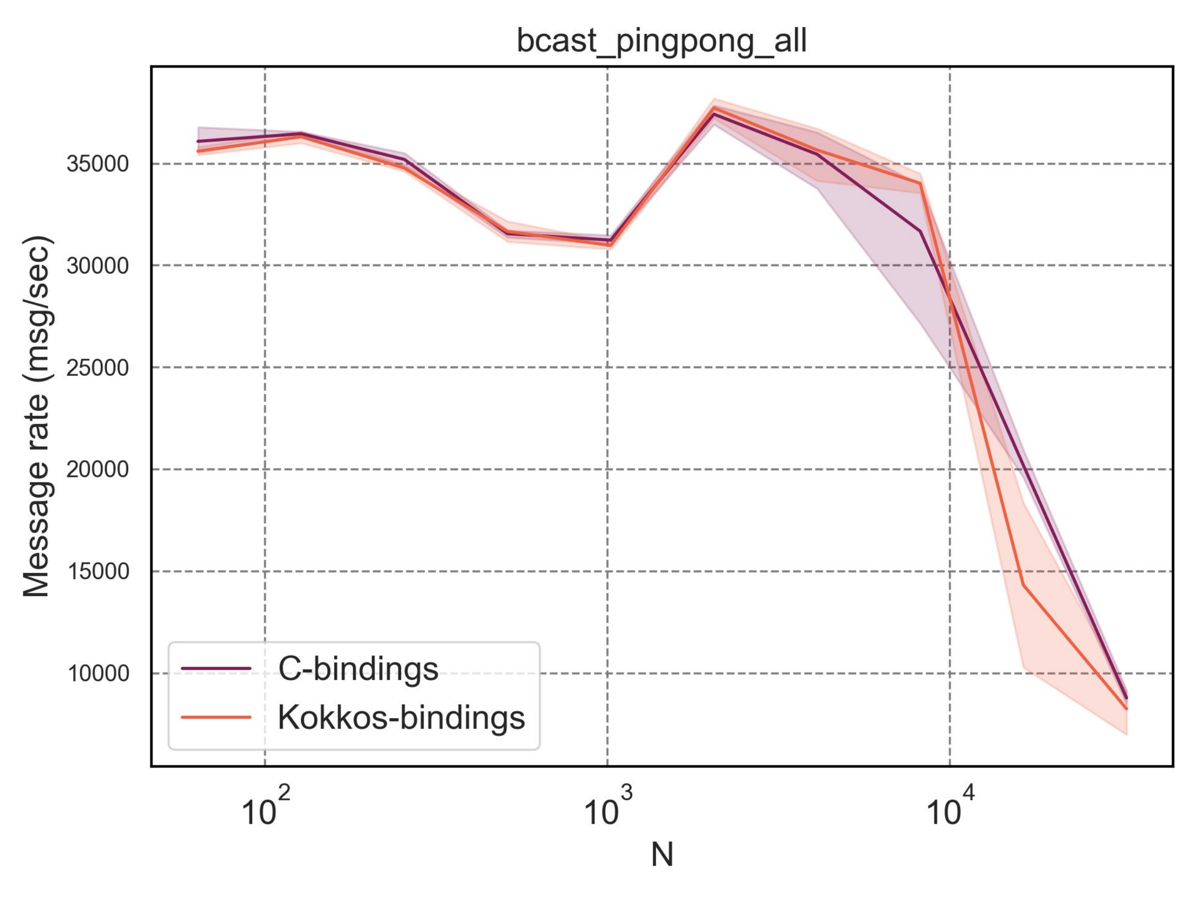


Three-Dimensional Tests



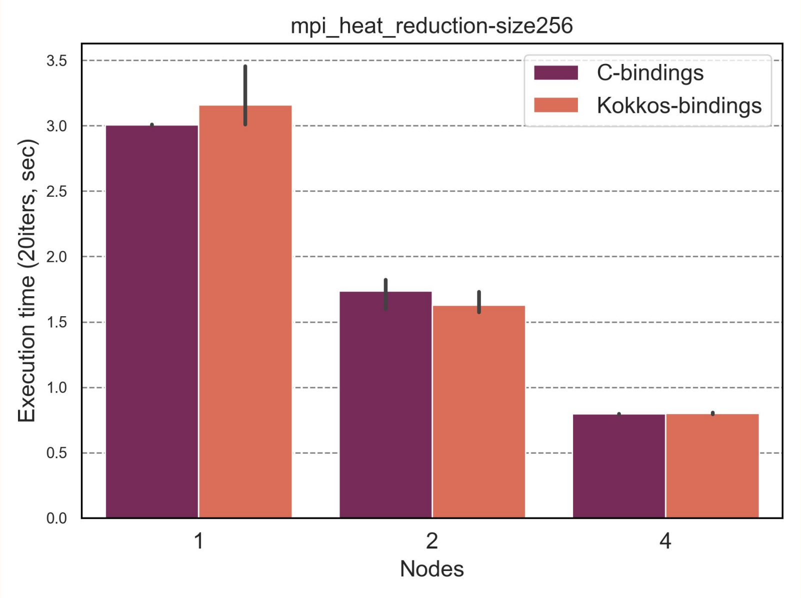
Broadcast Tests

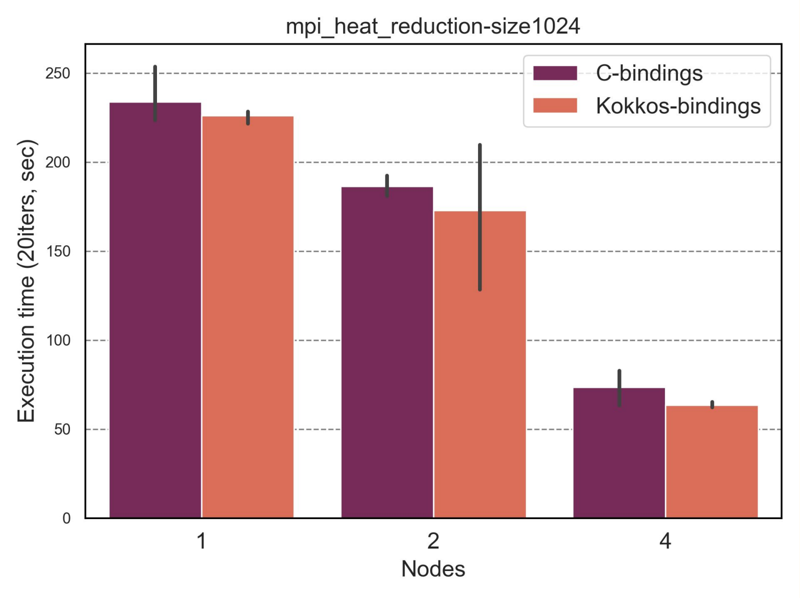
- Each test is run 100 times, then the first warm-up time is discarded.
- While the new bindings may look generally better here, all of this is more dependent on non-deterministic portions of timing and transports rather than simpler sends.



Heat Reduction Tests

- This test uses the heat reduction code in Kokkos tutorials which uses Views with MPI, with only the bindings altered for the our version.
- The Kokkos-bindings have a slightly better execution time, especially for the increased 1024 size





Conclusion

- This paper set about to integrate two programming models, MPI and Kokkos.
- Implemented several MPI bindings with Kokkos View objects as their primary buffers without sacrificing the C++ nature of the Kokkos View.
- Using ExaMPI benefitted the project, as it enables the use of templates to better interact with Kokkos.
- After implementing the bindings for this project, we found that the new bindings' performed similarly to the old bindings's.

Future Work

- Wider range of MPI functions such as All-To-All, Scatter, and Gather.
- Change from MPI_Kokkos_X to overloading existing MPI functions.
- More device-specific support (i.e., MPI_Send<View, class, Device>).
- Testbed for new functions, such as byte-mapping-based transports, rather than traditional datatype-based transports.
- Reconciling the differences between MPI and Kokkos methods of dealing with non-contiguous data, and add non-contiguous View support.
- A creation of new backends to increase speed for specific types of Views (i.e., Views on GPUs, non-contiguous, etc.).

Acknowledgements

Thanks to Riley Shipley and Derek Schafer for reviewing this work, along with the ExaMPI team and the Kokkos team for technical support. Additional thanks to Dr. Joseph Dumas II and Dr. Michael Ward.

Funding in part is acknowledged from these NSF Grants 191897, 21501020, and 2201497, as well as and the U.S. Department of Energy's National Nuclear Security Administration (NNSA) under the Predictive Science Academic Alliance Program (PSAAP-III), Award DE-NA0003966.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy’s National Nuclear Security Administration under contract DE-NA-0003525

Questions