Kokkos-Enhanced ExaMPI: Modern Parallel Programming for Exascale

Evan Drake Suggs

University of Tennessee at Chattanooga

Center for Understandable Performant Exascale Communication Systems

THE UNIVERSITY OF TENNESSEE CHATTANOOGA

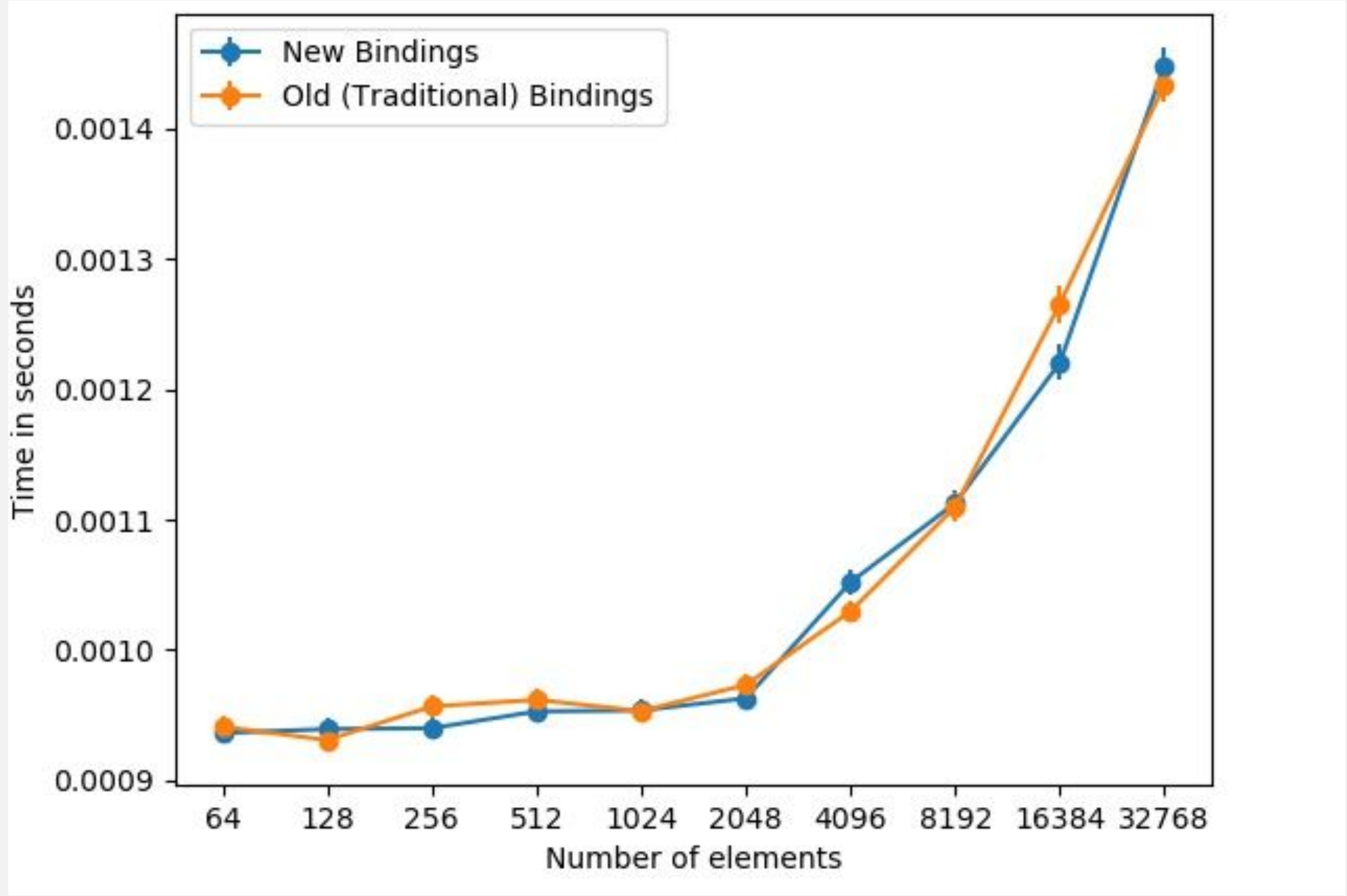
Kokkos-Enhanced ExaMPI: Modern Parallel Programming for Exascale

Evan Drake Suggs[1], Derek Schafer[2], and Anthony Skjellum[1] 1. UTC 2. UNM| UTC Computer Science

Background

- The Kokkos programming library provides in-memory advanced data structures, concurrency, and algorithms to support advanced C++ parallel programming while MPI provides a widely used message passing model for inter-node communication [1,2,4].
- The primary feature of Kokkos discussed in this poster is its implementation of the View, a datatype similar to a tensor [1]. A key feature of Kokkos is that Views can be assigned to specific memory and execution spaces [1, 2].
- The purpose of this poster is to integrate Kokkos within the ExaMPI implementation via a MPI Extension to obtain development benefits over having the two running separately in the same program. This is both for MPI itself and for applications that use MPI+Kokkos. First-class Kokkos objects can be passed directly to the new functions.
- The primary advantage will be the time saved during development rather than at runtime, as the underlying model is the same.
- This work has implications separate from Kokkos, such as how true MPI-based C++ bindings differ from classic C bindings.
- Previous work combining both Kokkos and traditional versions of MPI has yielded interesting results. For example, Khuvis et al. [2] have a shown a speedup of GEMM code and the Graph500 benchmark using their version of MPI+Kokkos.

Results



Times (in milliseconds) and their standard deviations for a traditional MPI+Kokkos Send/Recv versus the New bindings

The above tests consist of runs for two MPI ping-pong tests, one that manually sends a View using traditional MPI_Send/Recv to send the array and the other which uses the Kokkos-integrated bindings. These tests were run on a node of the EPYC cluster at the UTC Simcenter [5]. Each ping-pong test is run 101 times, with length ranging in size from 64 to 32768 elements. The time recorded for each ping-pong test is the completion time of the process with the first send.

Which bindings has better performance varies on almost every point. This is likely due to the run-time environment and access to resources rather than the code itself as there are no specific handling cases in the added code.

Overall, these results seem to be comparable for both methods with both types of bindings being better at different times and a very low standard deviation of the mean. Since the goal was roughly equal performance, this means that the bindings performed well.

Methods

```
// old method
    int *recv_buf = (int*) malloc(n * sizeof(int));
    MPI_Recv(recv_buf, n, MPI_INT, 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    Kokkos::View<int*> recv_check(recv_buf, n);
// new method
    Kokkos::View<int*> A("New Method View", n);
    MPI_Kokkos_Recv<Kokkos::View<int*>, int>(A, n, MPI_INT, 1, 0, MPI_COMM_WORLD);
// newer method
    Kokkos::View<int*> A("New Method View", n);
    MPI::Recv<Kokkos::View<int*>, int>(A, 1, 0, MPI_COMM_WORLD);
```

A comparison of existing and improved methods for sending the contents of a Kokkos View using MPI

- In the new method, all that is required to receive a View is to create an existing View of the correct size and copy to it.
- In the old method, the View's underlying pointer must be accessed by the programmer, opening up the possibility for memory leaks
- MPI_Kokkos_Send sends the underlying array as a Payload. Its counterpart, MPI_Kokkos_Recv receives this Payload, then wraps it back into a View object (either a View pointer or copying to full array with a performance penalty).
- This model is used for all the other MPI bindings as well. Note that this method is only compatible with contiguous arrays currently.
- Most of the bindings follow a traditional MPI binding except MPI_Kokkos_Get_Dims which returns every extent (size of each individual dimension) of the View as a single integer vector.
- The newer method no longer requires the MPI Datatype nor dimensions from the user.

Conclusions

This project integrated MPI and Kokkos by implementing several MPI bindings to use Kokkos Views as their primary buffers, keeping the C++ nature of the Kokkos View for users.
After implementing the bindings for this project, we found that the new bindings' performed similarly to the old bindings's.

- 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>).
- 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 Views
- Testbed for new functions, such as byte-mapping-based transports, rather than traditional datatype-based transports.

References

[1] H. Carter Edwards and Christian R. Trott. 2013. Kokkos: Enabling Performance Portability Across Manycore Architectures. In 2013 Extreme Scaling Workshop (xsw 2013). 18–24. https://doi.org/10.1109/XSW.2013.7
[2] Samuel Khuvis, Karen Tomko, Jahanzeb Hashmi, and Dhabaleswar K. Panda. 2020. Exploring Hybrid MPI+Kokkos Tasks Programming Model. In 2020 IEEE/ACM 3rd Annual Parallel Applications Workshop: Alternatives To MPI+X (PAW-ATM). 66–73. https://doi.org/10.1109/PAWATM51920.2020.00011
[3] Anthony Skjellum, Martin Rüfenacht, Nawrin Sultana, Derek Schafer, Ignacio Laguna, and Kathryn Mohror. 2020. ExaMPI: A Modern Design and Implementation to Accelerate Message Passing Interface Innovation. In High Performance Computing, Juan Luis Crespo-Mariño and Esteban Meneses-Rojas (Eds.). Springer International Publishing, Cham, 153–169.
[4] Christian R. Trott, Damien Lebrun-Grandié, Daniel Arndt, Jan Ciesko, Vinh Dang, Nathan Ellingwood, Rahulkumar Gayatri, Evan Harvey, Daisy S. Hollman, Dan Ibanez, Nevin Liber, Jonathan Madsen, Jeff Miles, David Poliakoff, Amy Powell, Sivasankaran Rajamanickam, Mikael Simberg, Dan Sunderland, Bruno Turcksin, and Jeremiah Wilke. 2022. Kokkos 3: Programming Model Extensions for the Exascale Era. IEEE Transactions on Parallel and Distributed Systems 33, 4 (2022), 805–817. https://doi.org/10.1109/TPDS.2021.3097283
[5] Epyc cluster [SimCenter wiki]. University of Tennessee at Chattanooga. Accessed February 13, 2023. https://wiki.simcenter.utc.edu/doku.php/clusters:epyc.

Acknowledgements

Thanks to Riley Shipley for reviewing this poster. We also thank the thesis committee, Dr. Joseph Dumas III and Dr. Michael Ward.

Funding

NSF Grants: #1821431, 1822191, 1918987, 1925603, 2201497