KokkosComm: Communication Layer for Distributed Kokkos Applications

Gabriel Dos Santos[1,2(✉)], Nicole Avans[3,4], Cédric Chevalier[1,2], Hugo Taboada[1,2], Carl W. Pearson[3], Jan Ciesko[3], Stephen L. Olivier[3], and Marc Pérache[1,2]

[1] CEA, DAM, DIF, F-91297 Arpajon, France
{gabriel.dossantos,cedric.chevalier,hugo.taboada,marc.perache}@cea.fr
[2] Université Paris-Saclay, LIHPC, France
[3] Sandia National Laboratories, Albuquerque, NM, USA
{cnavans,jciesko,slolivi,cwpears}@sandia.gov
[4] Tennessee Technological University, Cookeville, TN, USA

Abstract. The Kokkos C++ Performance Portability Programming Ecosystem provides multi-dimensional data structures, concurrency, and algorithms to support shared-memory heterogeneous programming on modern HPC architectures.
HPC applications that use Kokkos for intra-node parallelism typically rely on MPI to distribute their code across multiple nodes. For this kind of distributed application, programmers have to implement the logic to exchange Kokkos Views using MPI. In this short-paper, we introduce KokkosComm: an experimental, zero-overhead interface built on top of standard message-passing libraries. KokkosComm unifies existing practices by offering a streamlined API for exchanging first-class Kokkos objects within modern C++ applications.

Keywords: Kokkos · MPI · C++ · Heterogeneous Programming · High Performance Computing

1 Introduction

The Kokkos C++ Performance Portability Programming Ecosystem [5] is a set of libraries and a programming model providing shared-memory parallelism across CPU and GPU architectures. Kokkos' design philosophy is to offer simple core primitives (`parallel_for`, `parallel_reduce` and `parallel_scan`) that let users express parallel computations in a portable way with regards to the target hardware. It also introduces opaque data structures that model where data is stored and how it is laid out in memory. Expressing algorithms using Kokkos' primitives and data structures ensures performance portability across various hardware architectures. Moreover, the minimal nature of Kokkos' API discourages esoteric code patterns and may enable optimizations that could not be achieved in the general case, further helping to improve performance.

One of Kokkos's primary data structures is the View: a multi-dimensional array. Views may be sliced into subviews, potentially implying accessing data

elements not contiguous in memory. Exchanging Views in a distributed application implies different possible communication strategies. For instance, a non-contiguous View may be passed as multiple smaller, contiguous messages, or it may also be packed as a single, larger message.

Kokkos does not provide primitives for distributing computations across multiple nodes. To scale Kokkos applications, programmers must rely on message-passing frameworks such as MPI and write the interfaces for communicating Kokkos Views through these libraries. Currently, these application-specific wrappers are not unified across the Kokkos ecosystem, which leads to unnecessary code duplication.

Efforts like Teuchos [3] MPI, part of the Trilinos [4] project (from which Kokkos also stems), have tried providing abstractions over MPI but did not focus on the particularities of Kokkos objects. Attempts to integrate first-class support for Kokkos Views, as seen in ExaMPI[2] — a non-standard C++-based MPI implementation — have paved the way for more seamless interactions between these frameworks by integrating support for transferring Views directly into the ExaMPI implementation. However, the need for a unified, efficient approach based on standard MPI implementations persists, driving the exploration of new solutions to enhance performance portability in HPC applications based on Kokkos.

This short-paper introduces KokkosComm, a message-passing layer that simplifies building efficient, distributed Kokkos applications, seeking to address the following points:

1. Make Kokkos + MPI interoperability less error-prone by helping manage Kokkos parallel executions and MPI non-blocking operations;
2. Automatically choose the appropriate method for exchanging Views (non-contiguous data, non-GPU-aware MPI implementation, etc.);
3. Develop and evaluate performance-portable communication interfaces using the ecosystem developed by the points above and implemented using a variety of underlying communication libraries.

2 KokkosComm: toward a unified Kokkos + Message-Passing wrapper

Currently, application developers that rely on Kokkos and message-passing must implement bindings to manage their interaction themselves, which involves handling non-GPU-aware MPI implementations (e.g., when the data to communicate resides in a device's memory) and how to manage non-contiguous data (e.g., when exchanging a slice of a multi-dimensional View).

The primary goal of KokkosComm is to ease the use of explicit message-passing communications for Kokkos users. KokkosComm is aiming at being

performance-portable. For example, it automatically handles GPU and non-GPU-aware MPI implementations. It also provides efficient implementations of communication patterns that require special handling, such as exchanging non-contiguous Views. To this end, we follow Kokkos' design philosophy and focus on providing reliable core primitives to enable performance portability in distributed computing. We aim to unify existing practices as they exist in wrappers for current Kokkos + MPI applications and leverage optimization enabled by C++ meta-programming techniques and Kokkos Views' compile-time information (i.e., memory space and layout). Furthermore, the minimal API design promotes extending support for other MPI-like libraries, such as NCCL or other message-passing frameworks, in the future.

The KokkosComm project primarily focuses on MPI integration. At its core, we introduce an API that balances simplicity and functionality. By maintaining a minimal design philosophy, KokkosComm aims to reduce programming complexity while offering compelling support for distributed communications in Kokkos-based HPC environments. We implement a reduced subset of MPI's point-to-point and collective functions, focusing on non-blocking operations. Another vital aspect of this effort is ensuring direct interoperability when interacting with standard MPI objects (e.g., communicators or requests). It guarantees that KokkosComm is easy to integrate into applications that already use MPI or that are not purely Kokkos-based (e.g., that call external libraries expecting raw MPI objects).

Our effort also brings the opportunity to think about distributed communication semantics thoroughly. It lets us ensure that the proposed API accurately reflects and efficiently handles the nuances of inter-node data transfers within the context of heterogeneous computing. For instance, our high-level approach ensures we uphold the semantics of Kokkos and the various message-passing libraries we seek to support. Currently, an effort within the project tries to bring support for NVIDIA's NCCL usable through the same core API as we provide for MPI.

This dual abstraction and direct integration approach positions KokkosComm as a versatile solution that can adapt to evolving HPC communication needs while providing immediate, practical benefits to current MPI-based Kokkos applications.

3 Future work

The development of KokkosComm opens up exciting avenues for future research and innovation in message-passing frameworks for C++. One promising direction is the potential to leverage this work as a springboard for designing MPI-C++ extensions inspired by projects like ExaMPI. This greenfield effort could rely on novel language features of the latest C++ standards, aligning the API more closely with the evolving standards and best practices.

For instance, the upcoming ISO C++26 standard will introduce a senders/receivers model through `std::execution`[1]. This asynchronous programming paradigm could help us rethink how distributed MPI communications are written in C++. Integrating these concepts into an MPI-C++ interface could lead to more expressive and efficient ways of describing complex communication patterns and data dependencies in distributed applications.

Another area of focus would be the integration of C++23's `std::mdspan`[6], a standard multi-dimensional array view (closely related to Kokkos Views). This feature could provide a more natural and efficient way to represent and manipulate multi-dimensional data structures within the MPI context, potentially simplifying code and improving performance.

We aim to enhance KokkosComm by pursuing these directions and contribute to the broader evolution of message-passing-based programming models.

4 Conclusion

In this short-paper, we describe KokkosComm, an ongoing effort addressing the challenges of integrating MPI within Kokkos-based applications for efficient distributed computing. We propose a simple, generic, high-level API that follows Kokkos' design philosophy and enables writing performance-portable communication patterns in C++ using mainstream message-passing frameworks such as MPI or NCCL. Our approach also encourages collaborators to use the project's API as a research platform to integrate experimental distributed communication models into the Kokkos ecosystem. Application developers have started replacing their hand-written MPI interoperability layers with KokkosComm, which provides us with essential feedback for ongoing enhancements. Current efforts adding support for other message-passing libraries — particularly for NCCL/RCCL — through our high-level API will let Kokkos application developers trivially experiment with different communication back-ends.

Acknowledgments. This work was supported by the French Alternative Energies and Atomic Energy Commission (CEA).

This work was supported by the Predictive Science Academic Alliance Program (PSSAP), sponsored by the U.S. Department of Energy's National Nuclear Security Administration. 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. s

References

1. Dominiak, M., Evtushenko, G., Baker, L., Radu Teodorescu, L., Howes, L., Shoop, K., Garland, M., Niebler, E., Adelstein Lelbach, B.: P2300r10: `std::execution`. https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2024/p2300r10.html (2024)

2. Suggs, E., Olivier, S., Ciesko, J., Skjellum, A.: View-aware Message Passing Through the Integration of Kokkos and ExaMPI. In: Proceedings of the 30th European MPI Users' Group Meeting. EuroMPI '23, Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3615318.3615321, https://doi.org/10.1145/3615318.3615321
3. Teuchos Project Team, T.: The Teuchos Project Website, https://trilinos.github.io/teuchos.html
4. Trilinos Project Team, T.: The Trilinos Project Website (2020), https://trilinos.github.io
5. Trott, C., Berger-Vergiat, L., Poliakoff, D., Rajamanickam, S., Lebrun-Grandie, D., Madsen, J., Al Awar, N., Gligoric, M., Shipman, G., Womeldorff, G.: The Kokkos EcoSystem: Comprehensive Performance Portability for High Performance Computing. Computing in Science Engineering **23**(5), 10–18 (2021). https://doi.org/10.1109/MCSE.2021.3098509
6. Trott, C., Hollman, D.S., Lebrun-Grandie, D., Hoemmen, M., Sunderland, D., Edwards, H.C., Adelstein Lelbach, B., Bianco, M., Sander, B., Ilopoulos, A., Michopoulos, J., Liber, N.: P0009r18: MDSPAN. https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2022/p0009r18.html (2022)