Kokkos Comm

Performance Portable Communication Interface for Distributed Kokkos Applications

Gabriel Dos Santos[1,2], Hugo Taboada[1,2], Cédric Chevalier[1,2] & Marc Pérache[1,2]

[1] CEA, DAM, DIF, F-91297 Arpajon, France
[2] Université Paris-Saclay, CEA, LiHPC

Kokkos Tea Time – 2026-01-21

Disclaimer

Thanks to all the Kokkos Comm contributors!

CEA

- Cédric Chevalier
- Hugo Taboada
- Marc Pérache

Sandia National Labs

- Carl Pearson
- Stephen Olivier
- Matthew Dosanjh
- Vivek Kale
- Jan Ciesko

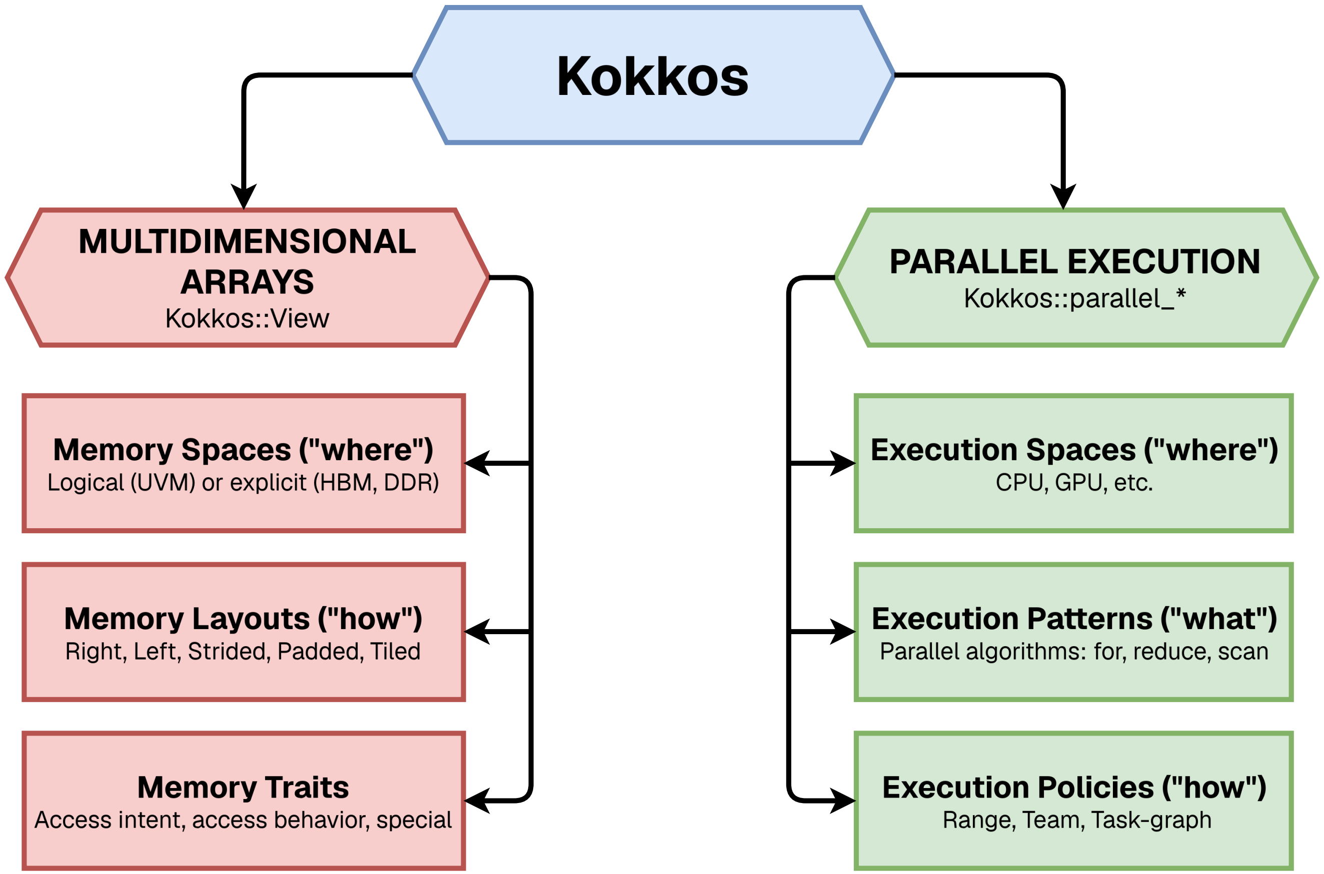
Tennessee Tech

- Nicole Avans
- Evan Suggs
- Anthony Skjellum

Contents

1. Context

Kokkos programming model



Distributed computing with Kokkos

Today:

- No “native” support for explicit communications
  - Hand-rolled interoperability
  - Generally with MPI
- Kokkos <=> MPI interfacing logic unique to each project
  - Code duplication across the Kokkos ecosystem
  - Different approaches and implementations to the handling of technical challenges and special cases

Distributed computing with Kokkos

Today:

- No "native" support for explicit communications
  - Hand-rolled interoperability
  - Generally with MPI
- Kokkos <=> MPI interfacing logic unique to each project
  - Code duplication across the Kokkos ecosystem
  - Different approaches and implementations to the handling of technical challenges and special cases

Technical Challenges:

How to pass Kokkos Views to MPI?

- How to handle non-contiguous data?
  - Packing & unpacking of one "big" contiguous packet
  - Use custom `MPI_Datatype`s
  - Send multiple "small" but contiguous data packets
- What if the MPI implementation isn't GPU-aware?
  - Host staging of device-stored Kokkos Views

How to deal with Kokkos Views lifetimes?

- What is the interaction between Kokkos & MPI?
- How to make asynchronous parallel computations and non-blocking communications operating on the same data consistent?

Example wrapping MPI_Isend

```cpp
template <typename E, typename V>
auto isend_wrapper(
  const E& exec,
  const V& view,
  int dst,
  int tag,
  MPI_Comm comm
) -> MPI_Request {
  using T = typename V::non_const_value_type;
  MPI_Datatype dtype = []() {
    if constexpr (is_same_v<T, int>) { return MPI_INT; }
    else if constexpr (...) { return ...; }
  }();
  MPI_Request req;
  using M = typename V::memory_space;
  if constexpr (!Kokkos::SpaceAccessibility<Kokkos::HostSpace, M>::accessible
                or !is_MPI_GPU_aware())
  {
    // Host staging if MPI isn't GPU-aware...
  } else {
    if (!v.span_is_contiguous()) {
      // Process non-contiguous views...
    } else {
      // Sync: wait for ongoing work in the Kokkos exec space
      ex.fence();
      // Extract view pointer and span to call MPI
      MPI_Isend(sv.data(), sv.span(), dtype, dst, tag, comm, &req);
    }
  }
  return req;
}
```

(1) View value type deduction

(2) Non-GPU-awareness handling

(3) Non-contiguous data handling

(4) Happy path to MPI call

2. Design

Challenges

- How to expose an API that allows Kokkos <=> MPI interoperability?
  - Handles MPI GPU-awareness
  - Handles non-contiguous data
  - Facilitates Views lifetimes management
  - Introduces minimal to no overhead
- Can we also design it to work with other communication libraries?
  - GPGPU/AI-oriented: NCCL, RCCL, oneCCL, etc.
  - Closer to the hardware/interconnects: libfabric, OFI, UCX, Portals4, etc.

Objectives

1. Design a high-level, compact, composable and extensible API
   - Abstract the concepts of the underlying communication libraries
   - Handle special cases
   - Minimize the interface: expose only a subset of P2P and Colls

     $\Longrightarrow$ *Easy-to-use, Hard-to-misuse*

Objectives

1. Design a high-level, compact, composable and extensible API
   - Abstract the concepts of the underlying communication libraries
   - Handle special cases
   - Minimize the interface: expose only a subset of P2P and Colls

   $\Longrightarrow$ *Easy-to-use, Hard-to-misuse*

2. Leverage innovations in recent standards revisions:
   - C++ language:
     - ISO C++20 advancements: compile-time guarantees, meta-programming techniques, error handling, etc.
     - Additions to the standard library: `std::mdspan`, `std::execution`, `std::expected`, etc.
   - Communication libraries:
     - MPI-specific: sessions, persistent/partitioned comms, etc.
     - Common between libs: stream-triggering, device-initiation, etc.

Objectives

1. Design a high-level, compact, composable and extensible API
   - Abstract the concepts of the underlying communication libraries
   - Handle special cases
   - Minimize the interface: expose only a subset of P2P and Colls

   $\Longrightarrow$ *Easy-to-use, Hard-to-misuse*

2. Leverage innovations in recent standards revisions:
   - C++ language:
     - ISO C++20 advancements: compile-time guarantees, meta-programming techniques, error handling, etc.
     - Additions to the standard library: `std::mdspan`, `std::execution`, `std::expected`, etc.
   - Communication libraries:
     - MPI-specific: sessions, persistent/partitioned comms, etc.
     - Common between libs: stream-triggering, device-initiation, etc.

3. Maintain performance portability
   - Complex systems: CPUs, multi-GPUs, multi-NICs
   - Different interconnect technologies (BXI, Slingshot, Infiniband, etc.)
   - Prepare for smart NICs

NCCL as example

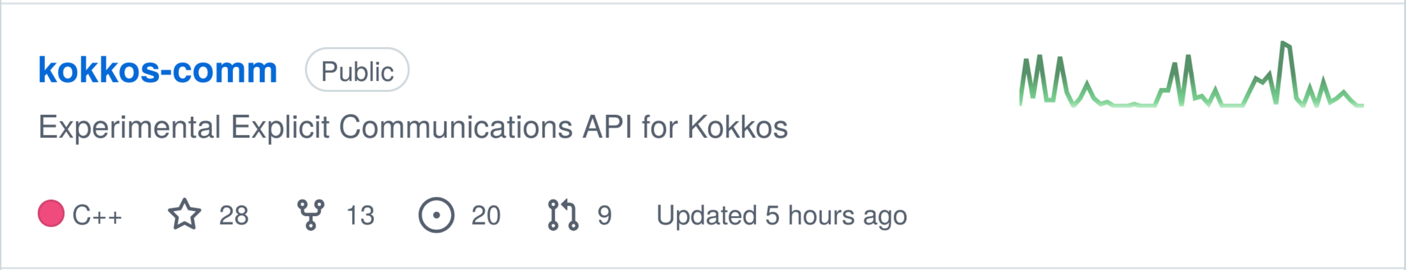
The NVIDIA Collective Communication Library

- Message passing communication library designed for NVIDIA GPUs
  - Written in C99-style C++
- Optimized for *device-to-device* communications
  - In particular for ML/AI patterns (e.g. Reduce-Scatter)
- API based on MPI, but highly streamlined:
  - Implemented as CUDA kernels
    - Synchronization based on CUDA streams/events
  - Designed to be collectives-first:
    - `Broadcast`, `All-Gather`, `All-Reduce`, `Reduce-Scatter` and `Reduce`
    - `Gather`, `Scatter` and `All-to-All` (since v2.28/Sep 2025)
  - Minimal point-to-point support: `Send`, `Recv`
  - Limited data type support
    - No FP complex, no custom data types
    - But support for reduced FP precisions: FP16, BF16, FP8 (E4M3 & E5M2)

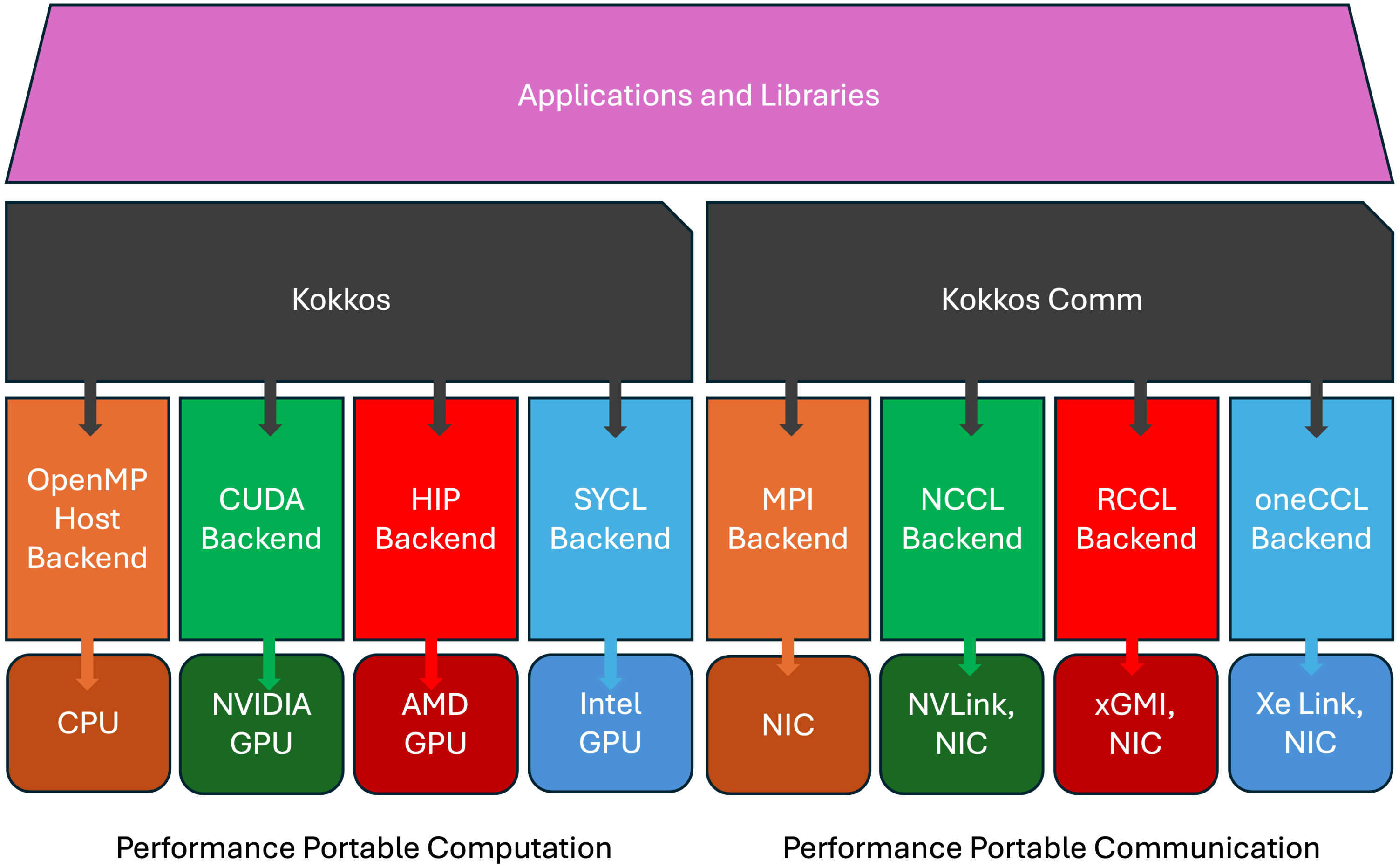
3. Kokkos Comm

Development framework

- Project debuted in March 2024
- Collaboration between CEA, Sandia National Laboratories and Tennessee Technological University
  - Other institutions participating sporadically (ORNL, CERFACS, UNM, etc.)
- Project hosted by the Kokkos organization
  - Member of the HPSF and Linux Foundation
  - Contributions open on GitHub: github.com/kokkos/kokkos-comm
- Dedicated discussion channels
  - Kokkos Slack `#mpi-interop` channel
  - GitHub Discussions and Issues pages
  - Bi-weekly developper meetings on HPSF Zoom (Thursdays, 16:30–17:00 CET / 8:30–9:00 MST)



The Kokkos stack



Technical choices (1/2)

- Model based on message-passing/two-sided
  - Explicit communications
  - cf. Kokkos Remote Spaces for one-sided models
- Written in C++20
  - Leverage compile-time metaprogramming to minimize overhead (`constexpr` all the things)
- Purely non-blocking/asynchronous API
  - Enable computation-communication overlap
  - Expose the asynchronism between the concurrent execution of compute and comms
- Forbid calling Kokkos Comm from within a parallel region (i.e., a kernel)
  - Cannot be supported for all backends/all Kokkos execution spaces
    - Cannot call MPI functions/NCCL kernels from a Kokkos parallel region running on device
  - Not detected by Kokkos Comm, UB if users do it anyway!

Technical choices (2/2)

- Implement the "high-level" logic
  - Resolve challenges related to managing special cases
    - GPU support (or absence of it) in communication libraries
    - Communication strategy depending on data layout
    - Managing/extending Views lifetimes to guarantee valid communications
  - Not a runtime
    - ✘ No re-implementing collectives algorithms
    - ✘ No low-level network management
    - Minimize overhead
  - Not a thin wrapper over MPI/NCCL/etc.
    - Identify the fundamental components common between communication libraries

      ✘ Don't want this

      ~~`send(KokkosView v, int count, MPI_Datatype dtype, int dst, int tag, MPI_Comm comm);`~~

      ✓ More like this

      ```
      send(CommHandle comm, KokkosView v, int dst);
      ```

Core API

Design

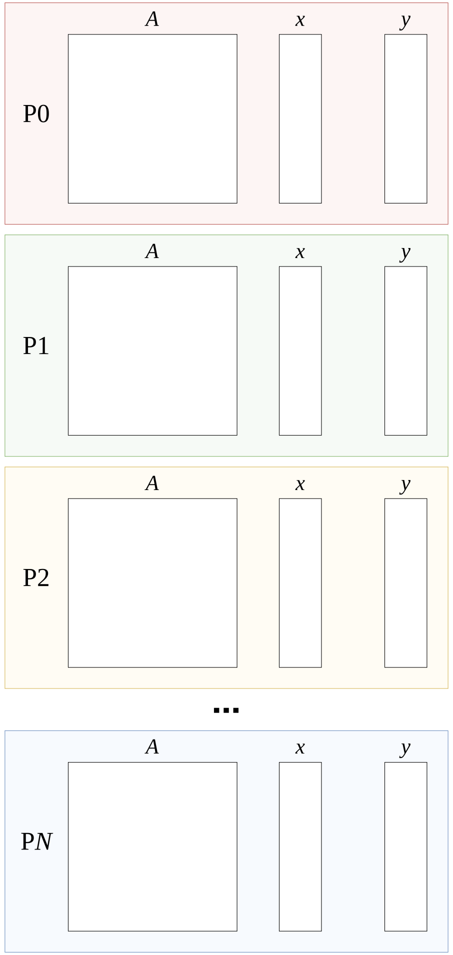
- Inspired by the minimalism of NCCL
- Built with C++20: modern metaprogramming techniques and standard library features
- Fully non-blocking interface: returns `Request` objects to complete operations

Overview

- 2 point-to-point:
  - `send`
  - `recv`
- 5 collectives:
  - `broadcast`
  - `all_gather`
  - `all_to_all`
  - `all_reduce`
  - `reduce`
- 2 synchronizations:
  - `wait`
  - `wait_all`

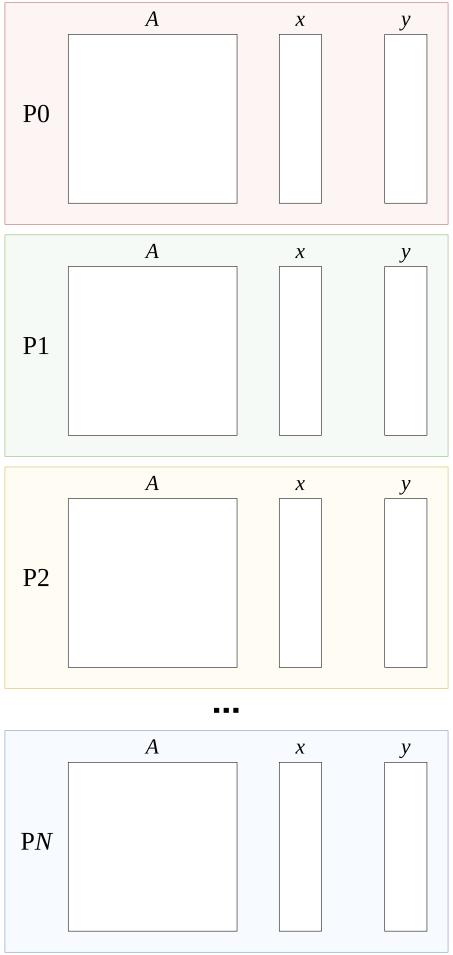
Example on distributed-GEMV

```cpp
// Handle wrapping Kokkos exec space + comm object (e.g. MPI_Comm, ncclComm_t, etc.)
KokkosComm::Handle handle(exec_space, comm_object);

Kokkos::View<double**> A("A", local_m, global_n);  // 1D row-wise partitioning
Kokkos::View<double*>  x("x", global_n);
Kokkos::View<double*>  y("y", local_m);

// Initialize local part of vector `x`
Kokkos::parallel_for("init local x",
  Kokkos::RangePolicy(exec_space, h.rank() * local_m, (h.rank() + 1) * local_m),
  KOKKOS_LAMBDA(int i) { x(i) = static_cast<double>(h.rank * i); });
exec_space.fence();

// Initiate in-place all-gather communication of `x`
auto allgather_req = KokkosComm::allgather(h, x, x);

// Initialize local part of `A` on each process
Kokkos::parallel_for("init local A",
  Kokkos::MDRangePolicy{0, 0}, {local_m, global_n}),
  KOKKOS_LAMBDA(int i, int j) { A(i, j) = static_cast<double>(h.rank() + j); }
);
exec_space.fence();

// Wait for all-gather of `x` to finish
KokkosComm::wait(allgather_req);

// Perform local matrix-vector multiplication
KokkosBlas::gemv("N", alpha, A, x, beta, y);
```



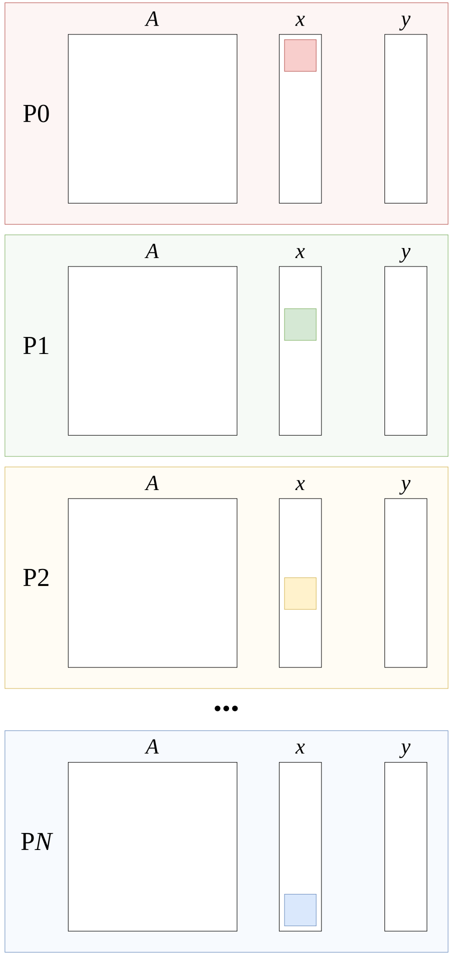
Example on distributed-GEMV

```cpp
// Handle wrapping Kokkos exec space + comm object (e.g. MPI_Comm, ncclComm_t, etc.)
KokkosComm::Handle handle(exec_space, comm_object);

Kokkos::View<double**> A("A", local_m, global_n);  // 1D row-wise partitioning
Kokkos::View<double*>  x("x", global_n);
Kokkos::View<double*>  y("y", local_m);

// Initialize local part of vector `x`
Kokkos::parallel_for("init local x",
  Kokkos::RangePolicy(exec_space, h.rank() * local_m, (h.rank() + 1) * local_m),
  KOKKOS_LAMBDA(int i) { x(i) = static_cast<double>(h.rank * i); });
exec_space.fence();

// Initiate in-place all-gather communication of `x`
auto allgather_req = KokkosComm::allgather(h, x, x);

// Initialize local part of `A` on each process
Kokkos::parallel_for("init local A",
  Kokkos::MDRangePolicy{0, 0}, {local_m, global_n}),
  KOKKOS_LAMBDA(int i, int j) { A(i, j) = static_cast<double>(h.rank() + j); }
);
exec_space.fence();

// Wait for all-gather of `x` to finish
KokkosComm::wait(allgather_req);

// Perform local matrix-vector multiplication
KokkosBlas::gemv("N", alpha, A, x, beta, y);
```



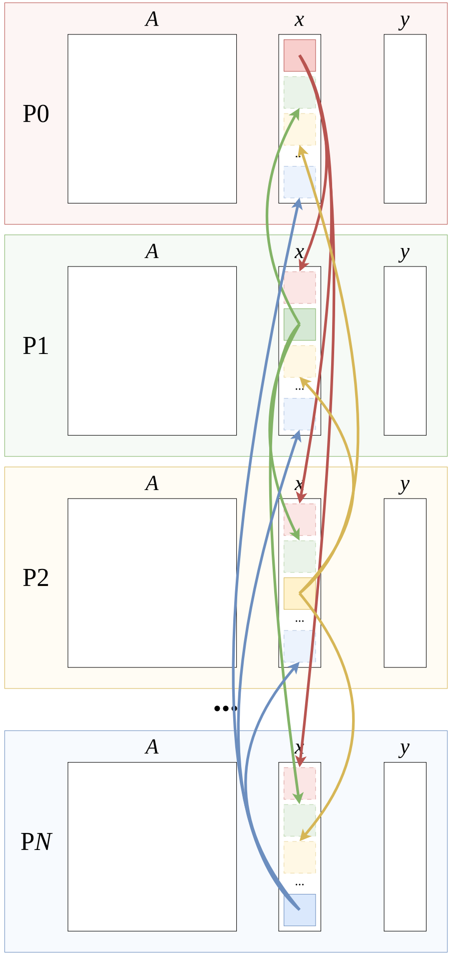
Example on distributed-GEMV

```cpp
// Handle wrapping Kokkos exec space + comm object (e.g. MPI_Comm, ncclComm_t, etc.)
KokkosComm::Handle handle(exec_space, comm_object);

Kokkos::View<double**> A("A", local_m, global_n);  // 1D row-wise partitioning
Kokkos::View<double*>  x("x", global_n);
Kokkos::View<double*>  y("y", local_m);

// Initialize local part of vector `x`
Kokkos::parallel_for("init local x",
  Kokkos::RangePolicy(exec_space, h.rank() * local_m, (h.rank() + 1) * local_m),
  KOKKOS_LAMBDA(int i) { x(i) = static_cast<double>(h.rank * i); });
exec_space.fence();

// Initiate in-place all-gather communication of `x`
auto allgather_req = KokkosComm::allgather(h, x, x);

// Initialize local part of `A` on each process
Kokkos::parallel_for("init local A",
  Kokkos::MDRangePolicy{0, 0}, {local_m, global_n}),
  KOKKOS_LAMBDA(int i, int j) { A(i, j) = static_cast<double>(h.rank() + j); }
);
exec_space.fence();

// Wait for all-gather of `x` to finish
KokkosComm::wait(allgather_req);

// Perform local matrix-vector multiplication
KokkosBlas::gemv("N", alpha, A, x, beta, y);
```



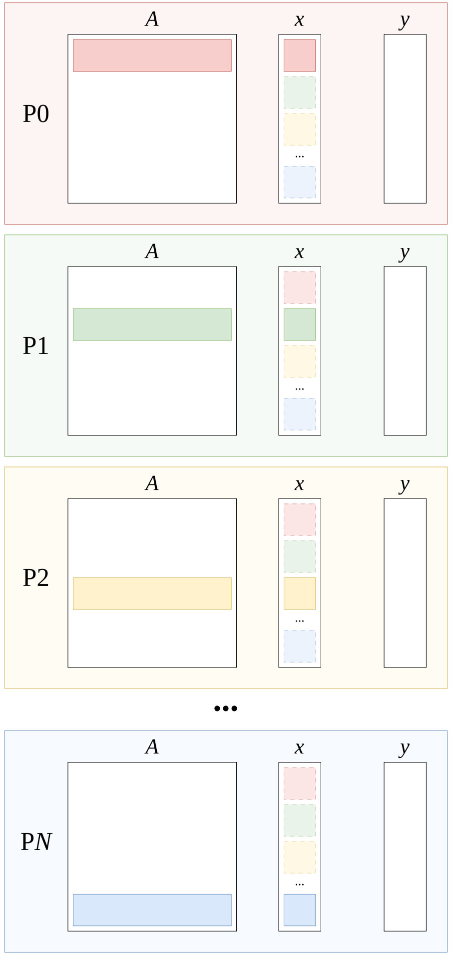
Example on distributed-GEMV

```cpp
// Handle wrapping Kokkos exec space + comm object (e.g. MPI_Comm, ncclComm_t, etc.)
KokkosComm::Handle handle(exec_space, comm_object);

Kokkos::View<double**> A("A", local_m, global_n);  // 1D row-wise partitioning
Kokkos::View<double*>  x("x", global_n);
Kokkos::View<double*>  y("y", local_m);

// Initialize local part of vector `x`
Kokkos::parallel_for("init local x",
  Kokkos::RangePolicy(exec_space, h.rank() * local_m, (h.rank() + 1) * local_m),
  KOKKOS_LAMBDA(int i) { x(i) = static_cast<double>(h.rank * i); });
exec_space.fence();

// Initiate in-place all-gather communication of `x`
auto allgather_req = KokkosComm::allgather(h, x, x);

// Initialize local part of `A` on each process
Kokkos::parallel_for("init local A",
  Kokkos::MDRangePolicy({0, 0}, {local_m, global_n}),
  KOKKOS_LAMBDA(int i, int j) { A(i, j) = static_cast<double>(h.rank() + j); }
);
exec_space.fence();

// Wait for all-gather of `x` to finish
KokkosComm::wait(allgather_req);

// Perform local matrix-vector multiplication
KokkosBlas::gemv("N", alpha, A, x, beta, y);
```



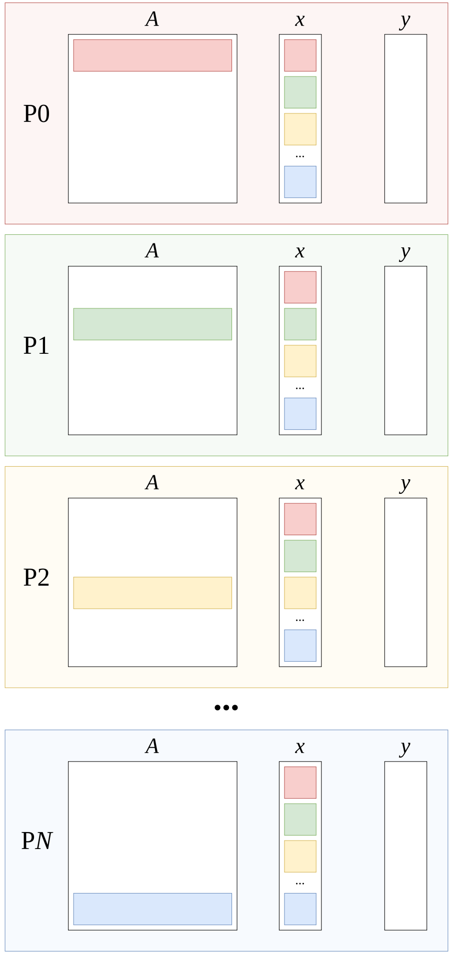
Example on distributed-GEMV

```cpp
// Handle wrapping Kokkos exec space + comm object (e.g. MPI_Comm, ncclComm_t, etc.)
KokkosComm::Handle handle(exec_space, comm_object);

Kokkos::View<double**> A("A", local_m, global_n);  // 1D row-wise partitioning
Kokkos::View<double*>  x("x", global_n);
Kokkos::View<double*>  y("y", local_m);

// Initialize local part of vector `x`
Kokkos::parallel_for("init local x",
  Kokkos::RangePolicy(exec_space, h.rank() * local_m, (h.rank() + 1) * local_m),
  KOKKOS_LAMBDA(int i) { x(i) = static_cast<double>(h.rank * i); });
exec_space.fence();

// Initiate in-place all-gather communication of `x`
auto allgather_req = KokkosComm::allgather(h, x, x);

// Initialize local part of `A` on each process
Kokkos::parallel_for("init local A",
  Kokkos::MDRangePolicy{0, 0}, {local_m, global_n}),
  KOKKOS_LAMBDA(int i, int j) { A(i, j) = static_cast<double>(h.rank() + j); }
);
exec_space.fence();

// Wait for all-gather of `x` to finish
KokkosComm::wait(allgather_req);

// Perform local matrix-vector multiplication
KokkosBlas::gemv("N", alpha, A, x, beta, y);
```



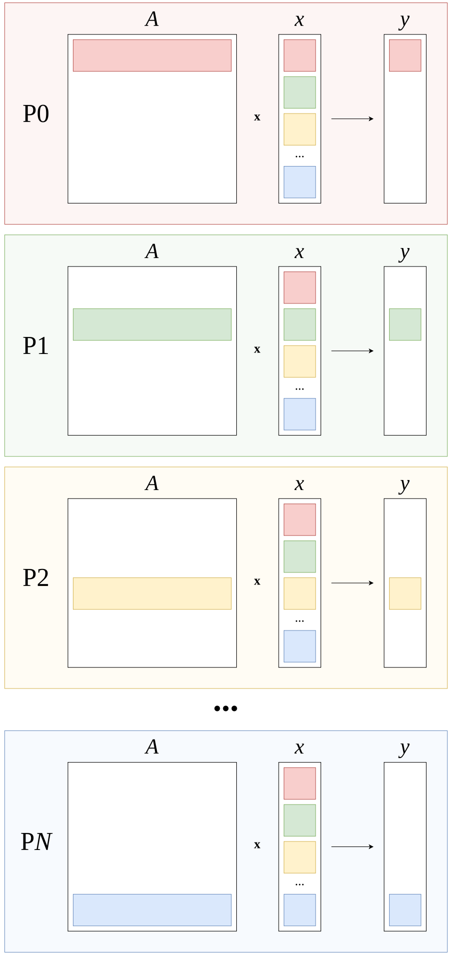
Example on distributed-GEMV

```
// Handle wrapping Kokkos exec space + comm object (e.g. MPI_Comm, ncclComm_t, etc.)
KokkosComm::Handle handle(exec_space, comm_object);

Kokkos::View<double**> A("A", local_m, global_n);  // 1D row-wise partitioning
Kokkos::View<double*>  x("x", global_n);
Kokkos::View<double*>  y("y", local_m);

// Initialize local part of vector `x`
Kokkos::parallel_for("init local x",
  Kokkos::RangePolicy(exec_space, h.rank() * local_m, (h.rank() + 1) * local_m),
  KOKKOS_LAMBDA(int i) { x(i) = static_cast<double>(h.rank * i); });
exec_space.fence();

// Initiate in-place all-gather communication of `x`
auto allgather_req = KokkosComm::allgather(h, x, x);

// Initialize local part of `A` on each process
Kokkos::parallel_for("init local A",
  Kokkos::MDRangePolicy{0, 0}, {local_m, global_n}),
  KOKKOS_LAMBDA(int i, int j) { A(i, j) = static_cast<double>(h.rank() + j); }
);
exec_space.fence();

// Wait for all-gather of `x` to finish
KokkosComm::wait(allgather_req);

// Perform local matrix-vector multiplication
KokkosBlas::gemv("N", alpha, A, x, beta, y);
```



Example on distributed-GEMV

```
// Handle wrapping Kokkos exec space + comm object (e.g. MPI_Comm, ncclComm_t, etc.)
KokkosComm::Handle handle(exec_space, comm_object);

Kokkos::View<double**> A("A", local_m, global_n);  // 1D row-wise partitioning
Kokkos::View<double*>  x("x", global_n);
Kokkos::View<double*>  y("y", local_m);

// Initialize local part of vector `x`
Kokkos::parallel_for("init local x",
  Kokkos::RangePolicy(exec_space, h.rank() * local_m, (h.rank() + 1) * local_m),
  KOKKOS_LAMBDA(int i) { x(i) = static_cast<double>(h.rank * i); });
exec_space.fence();

// Initiate in-place all-gather communication of `x`
auto allgather_req = KokkosComm::allgather(h, x, x);

// Initialize local part of `A` on each process
Kokkos::parallel_for("init local A",
  Kokkos::MDRangePolicy{0, 0}, {local_m, global_n}),
  KOKKOS_LAMBDA(int i, int j) { A(i, j) = static_cast<double>(h.rank() + j); }
);
exec_space.fence();

// Wait for all-gather of `x` to finish
KokkosComm::wait(allgather_req);

// Perform local matrix-vector multiplication
KokkosBlas::gemv("N", alpha, A, x, beta, y);
```



Supported Backends

MPI: `KokkosComm::MpiSpace`

- Supported in the core API
  - Missing GPU support for non-blocking reduction collectives
- Extensions
  - Comm modes: Standard, Synchronized, Ready
  - Channels
    - Persistent comms (Send/Recv only)
    - Partitioned comms, in development at TTU & UNM
  - Stream-triggering, in development at TTU & UNM

NCCL: `KokkosComm::NcclSpace`

- Contribution largely lead by CEA
- Stream-triggered: allows to bypass some explicit synchronizations by leveraging Kokkos execution spaces
- Available in the core API since late October
  - Behind the `Experimental` namespace
  - Functionnal but likely needs more testing

Feature		**Current Support**		
		Core API	**MPI backend**	**NCCL backend**
Point-to-point	Send	✓	✓	✓
	Recv	✓	✓	✓
Collectives	Broadcast	✓	✓	✓
	All-Gather	✓	✓	✓
	Reduce	✓	✓ (B)	✓
	All-Reduce	✓	✓ (B)	✓
	All-to-All	✓	✓	✓
	Inclusive Scan	✗	✓ (B)	✗
	Exclusive Scan	✗	✓ (B)	✗
	Barrier	✗	✓ (B)	✗
Persistent	Send-init	✗	✓	✗
	Recv-init	✗	✓	✗
	Start-all	✗	✓	✗
Packing Strategy	Temporary Buffers	✓	✓	✓
	Custom Data Types	✓	✓	✗
GPU Awareness	NVIDIA CUDA	✓	✓	✓
	AMD HIP	✓	✓	✗
	Host staging	✓	✓	✗

Legend: (B) = Blocking-only
All calls are non-blocking unless otherwise specified. The MPI backend also exposes blocking variants for all interfaces.

Ongoing work

- Non-contiguous data handling not yet fully implemented
  - Supported: `send`, `recv`, `reduce`
  - Not supported yet: `all_gather`, `all_reduce`, `all_to_all`, `broadcast`
- Views of rank > 1 are not allowed yet in collectives
  - PR opened to lift this constraint
  - Expected behavior:
    Same as the underlying backend: dismiss rank and let the communication library interpret the buffer as if it were 1D
- MPI Host staging is not yet "smart"
  - Stages views whenever they aren't accessible from the host
    - CMake option letting users specify that the provided MPI is GPU-aware
- Paper submitted at PASC 2026
  - Review notifications on February 4th

Kokkos Comm: Performance Portable Communication for Distributed Kokkos Applications

C. Nicole Avans
Sandia National Laboratories
Albuquerque, NM, USA
Tennessee Technological University
Cookeville, TN, USA
cnavans42@tntech.edu

Gabriel Dos Santos
CEA, DAM, DIF, F-91297
Arpajon, France
Université Paris-Saclay, CEA, LiHPC
France
gabriel.dossantos@cea.fr

Cédric Chevalier
CEA, DAM, DIF, F-91297
Arpajon, France
Université Paris-Saclay, CEA, LiHPC
France
cedric.chevalier@cea.fr

Hugo Taboada
CEA, DAM, DIF, F-91297
Arpajon, France
Université Paris-Saclay, CEA, LiHPC
France
hugo.taboada@cea.fr

Marc Pérache
CEA, DAM, DIF, F-91297
Arpajon, France
Université Paris-Saclay, CEA, LiHPC
France
marc.perache@cea.fr

Matthew G. F. Dosanjh
Sandia National Laboratories
Albuquerque, NM, USA
mdosanj@sandia.gov

Stephen L. Olivier
Sandia National Laboratories
Albuquerque, NM, USA
slolivi@sandia.gov

Carl Pearson
Sandia National Laboratories
Albuquerque, NM, USA
cwpears@sandia.gov

Vivek Kale
Sandia National Laboratories
Livermore, CA, USA
vlkale@sandia.gov

Evan D. Suggs
Tennessee Technological University
Cookeville, TN, USA
esuggs@tntech.edu

Anthony Skjellum
Tennessee Technological University
Cookeville, TN, USA
askjellum@tntech.edu

Abstract

This paper introduces Kokkos Comm, a new library and API specification designed to enhance performance, portability, and productivity. Kokkos is a widely used C++ performance portability ecosystem that addresses performance portability on-node through advanced C++ metaprogramming, including GPU and OpenMP targeting support. Kokkos Comm is designed to address challenges of integrating Kokkos with distributed memory programming models. In particular, Kokkos Comm helps alleviate accidental complexities associated with coordinating non-blocking communication operations and Kokkos execution spaces, as well as handling non-contiguous data structures represented by `Kokkos::Views`. Automation of common low-level, error-prone implementation details, such as packing and unpacking of non-contiguous data, increases programmer productivity and decreases code complexity with the added potential for higher performance than using either MPI derived datatypes or user-coded loops. Additionally, this library serves as a platform for researching improved methods of managing non-contiguous data and exploring new communication APIs with performance portability across various underlying transports and accelerators. Kokkos Comm also enables use of varied internal implementations of the data transfer functionality (e.g., MPI RMA, NCCL, SHMEM dialects, libfabric), while maintaining overall support for MPI elsewhere in an application (e.g., ScaLAPACK). The research underlying Kokkos Comm aims to provide a unified, productive, performance-portable model for on- and off-node parallel programming, with headroom for continued enhancements of performance over time, and interoperability with MPI programming in the large.

Keywords

HPC, Kokkos, C++, performance portability, programming models, distributed computing, communication

ACM Reference Format:
C. Nicole Avans, Gabriel Dos Santos, Cédric Chevalier, Hugo Taboada, Marc Pérache, Matthew G. F. Dosanjh, Stephen L. Olivier, Carl Pearson, Vivek Kale, Evan D. Suggs, and Anthony Skjellum. 2025. Kokkos Comm: Performance Portable Communication for Distributed Kokkos Applications. In *Proceedings of Platform for Advanced Scientific Computing Conference (PASC '26)*. ACM, New York, NY, USA, 12 pages. https://doi.org/XXXXXXX.XXXXXXX

PASC '26, Bern, Switzerland

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Increasingly complex high-performance computing (HPC) architectures have led to a demand for efficient integration of on-node and off-node performance-portable programming models. Kokkos [38],

4. Future Work

Other communication backends

Device/AI-oriented libraries

- AMD RCCL (ROCm Communication Collectives Library)
  - Identical API to NCCL: “simple” integration, reuse existing bindings
  - Exploration in progress
- Intel oneCCL (oneAPI Collective Communications Library)
  - Close to NCCL/RCCL
    - API is closer to MPI-style
    - Small interface that fits nicely with Kokkos Comm’s core API

Other communication backends

Device/AI-oriented libraries

- AMD RCCL (ROCm Communication Collectives Library)
  - Identical API to NCCL: “simple” integration, reuse existing bindings
  - Exploration in progress
- Intel oneCCL (oneAPI Collective Communications Library)
  - Close to NCCL/RCCL
    - API is closer to MPI-style
    - Small interface that fits nicely with Kokkos Comm’s core API

Interconnect/network-oriented libraries

- Portals4
  - Low-level API used in Atos BXI & HPE Slingshot
- MPC Low Comm
  - Module of MPC (Multi-Process Computing framework) an MPI+OpenMP implementation developed at CEA
  - Intermediate communication API between MPI and low-level network layers (similar to e.g. OFI)

Execution semantics (1/2)

Stream-triggering

Automatically launch communication operations based on resource availability on an associated computational stream:

- Allows certain explicit synchronizations to be eliminated
  $\Longrightarrow$ Improved coordination between Kokkos and the communication library (particularly MPI)
- Requires users to leverage Kokkos execution spaces

```
/*(1)*/   Kokkos::parallel_for(exec1, ... );

          // Stream-ordered: okay to use views produced in (1) without fence
/*(2)*/   auto req = KokkosComm::allgather(exec1, ... );

          // Executes concurrently with (2)
/*(3)*/   Kokkos::parallel_for(exec2, ... );

          // Ensure that the communication finishes
          // May need to deep copy non-contiguous receive views used in (2)
/*(4)*/   KokkosComm::wait(req);

          // OK to launch without fence, views updated in (2) are guaranteed to be ready because of (4)
/*(5)*/   Kokkos::parallel_for(exec1, ... );
```

Execution semantics (2/2)

`std::execution`: Senders/Receivers API in C++26

- Similar to CUDA Graphs
- Multiple reference implementations: `stdexec` (NVIDIA), `beman.execution` (Beman Standard), `pika` (HPX)
- Expressions of dependencies between operations (computations and communications) in the form of DAGs

```
MPI_Irecv( ... , &request);

kernel<<< ... , stream>>>( ... );
cudaEventRecord(event, stream);

MPI_Wait(&request, ... );
cudaEventSynchronize(event);
```

$\Longrightarrow$

```
sender auto s_recv = ... | transform_mpi(MPI_Irecv);

sender auto s_cuda = ... | then_with_stream(kernel);

sender auto s_cont = when_all(s_recv, s_cuda) | then( ... );
```

Example taken from CSCS's MPI and std::execution (the C++ 26 async API) slides

Other planned or ongoing efforts

Framework integration

- Integration of Kokkos Comm in Trilinos and Tpetra

Other planned or ongoing efforts

Framework integration

- Integration of Kokkos Comm in Trilinos and Tpetra

Overlap of communications & data transformations

- On-the-fly transformation of multi-dimensional data layouts
  - For example, transposing a 2D view into the optimal layout based on the architecture of the communication target
- Integration with optimized custom layouts (e.g. data tiling)

Other planned or ongoing efforts

Framework integration

- Integration of Kokkos Comm in Trilinos and Tpetra

Overlap of communications & data transformations

- On-the-fly transformation of multi-dimensional data layouts
  - For example, transposing a 2D view into the optimal layout based on the architecture of the communication target
- Integration with optimized custom layouts (e.g. data tiling)

Contribution to standards

- Serve as reference work for standardization efforts
  - Support for multidimensional data in distributed explicit communication libraries
  - Modern C++ API for MPI

5. Conclusion

Conclusion

What Kokkos Comm provides

A lightweight library providing Kokkos-native explicit communications for distributed applications

- Generic asynchronous API able to target multiple backend communication libraries
  - Support for MPI and NCCL, with more backends coming
  - Deep integration with Kokkos' execution model
- Minimal interface leveraging modern C++ to increase simplicity and usability
  - Designed to be *"Easy-to-use, Hard-to-misuse"*, with near-zero overhead

Handles Kokkos + communication libraries pain points

- GPU-awareness
  - Automatic host staging when backend does not support GPU-aware communications
- Non-contiguous data
  - Automatic marshalling of data when communicating non-contiguous Views/subviews
- Views lifetime management
  - Automatic lifetime extension to guarantee that communications don't fail on freed data

Kokkos Comm welcomes you!

If you are interested in contributing to the following topics

- Trying out Kokkos Comm in the real world
  - Write new Kokkos + Kokkos Comm apps/mini-apps
  - Port existing Kokkos+MPI codes
  - Are there obvious interfaces that are missing from Kokkos Comm that you absolutely need?

Kokkos Comm welcomes you!

If you are interested in contributing to the following topics

- Trying out Kokkos Comm in the real world
  - Write new Kokkos + Kokkos Comm apps/mini-apps
  - Port existing Kokkos+MPI codes
  - Are there obvious interfaces that are missing from Kokkos Comm that you absolutely need?
- Help design what's next for Kokkos Comm
  - Determine how we should tackle Initialize/Finalize
  - Explore other communication backends (RCCL, oneCCL, UCX, OFI, ...)
  - Design/improve error handling
  - Extend to support a Senders/Receivers-based interface

Kokkos Comm welcomes you!

If you are interested in contributing to the following topics

- Trying out Kokkos Comm in the real world
  - Write new Kokkos + Kokkos Comm apps/mini-apps
  - Port existing Kokkos+MPI codes
  - Are there obvious interfaces that are missing from Kokkos Comm that you absolutely need?
- Help design what's next for Kokkos Comm
  - Determine how we should tackle Initialize/Finalize
  - Explore other communication backends (RCCL, oneCCL, UCX, OFI, …)
  - Design/improve error handling
  - Extend to support a Senders/Receivers-based interface
- Help the development and improve the infrastructure of Kokkos Comm
  - Implement new, and enhance existing tests
  - Write benchmarks and code examples
  - Improve CI/CD & DevOps
  - Help with code reviews
- Documentation updating, writing, proof-reading, etc.

Join the conversation on Slack, GitHub, or at the developer meetings!

Links to join us

Slack https://kokkosteam.slack.com/ (`#mpi-interop` channel)

GitHub

- Repository https://github.com/kokkos/kokkos-comm
- Discussions https://github.com/kokkos/kokkos-comm/discussions
- Issues https://github.com/kokkos/kokkos-comm/issues

Developer Meeting https://zoom-lfx.platform.linuxfoundation.org/meeting/95426536306?password=fd73274e-8949-4c61-baa5-fe6e1b71e18f (bi-weekly on Thursdays, 16:30–17:00 CET / 8:30–9:00 MST)

Thank you for your attention

Questions?

CEA, DAM, DIF, F-91297 Arpajon, France
Gabriel Dos Santos
gabriel.dossantos@cea.fr

API Reference

Core API

```
template <Kokkos::ExecutionSpace E, KokkosComm::CommunicationSpace C, KokkosComm::KokkosView V>
auto KokkosComm::send(KokkosComm::Handle<E, C>& handle, const V& view, int dst) -> KokkosComm::Request<C>;

template <Kokkos::ExecutionSpace E, KokkosComm::CommunicationSpace C, KokkosComm::KokkosView V>
auto KokkosComm::recv(KokkosComm::Handle<E, C>& handle, V& view, int src) -> KokkosComm::Request<C>;

template <Kokkos::ExecutionSpace E, KokkosComm::CommunicationSpace C, KokkosComm::KokkosView V>
auto KokkosComm::broadcast(KokkosComm::Handle<E, C>& handle, V& view, int root) -> KokkosComm::Request<C>;

template <Kokkos::ExecutionSpace E, KokkosComm::CommunicationSpace C, KokkosComm::KokkosView SV, KokkosComm::KokkosView RV>
auto KokkosComm::all_gather(KokkosComm::Handle<E, C>& handle, const SV& send_view, RV& recv_view) -> KokkosComm::Request<C>;

template <Kokkos::ExecutionSpace E, KokkosComm::CommunicationSpace C, KokkosComm::KokkosView SV, KokkosComm::KokkosView RV>
auto KokkosComm::all_to_all(KokkosComm::Handle<E, C>& handle, const SV& send_view, RV& recv_view, int count) -> KokkosComm::Request<C>;

template <Kokkos::ExecutionSpace E, KokkosComm::CommunicationSpace C, KokkosComm::KokkosView SV, KokkosComm::KokkosView RV, KokkosComm::ReductionOperator R>,
auto KokkosComm::all_reduce(KokkosComm::Handle<E, C>& handle, const SV& send_view, RV& recv_view, [[maybe_unused]] RO red_op) -> KokkosComm::Request<C>;

template <Kokkos::ExecutionSpace E, KokkosComm::CommunicationSpace C, KokkosComm::KokkosView SV, KokkosComm::KokkosView RV, KokkosComm::ReductionOperator R>,
auto KokkosComm::reduce(KokkosComm::Handle<E, C>& handle, const SV& send_view, RV& recv_view, int root, [[maybe_unused]] RO red_op) -> KokkosComm::Request<C>;

template <KokkosComm::CommunicationSpace C>
auto KokkosComm::wait(Request<C>& req) -> void;

template <KokkosComm::CommunicationSpace C>
auto KokkosComm::wait_all(std::span<Request<C>> req) -> void;

template <KokkosComm::CommunicationSpace C, typename T>
[[nodiscard]] constexpr auto datatype() -> typename C::datatype_type;
template <KokkosComm::CommunicationSpace C, typename T>
[[nodiscard]] constexpr auto datatype_for(T&&) -> typename C::datatype_type;
template <KokkosComm::CommunicationSpace C, KokkosComm::KokkosView V>
[[nodiscard]] constexpr auto datatype_for_view(V&&) -> typename C::datatype_type;

template <CommunicationSpace C, ReductionOperator R>
[[nodiscard]] constexpr auto reduction_op() -> typename C::reduction_op_type;
template <CommunicationSpace C, ReductionOperator R>
[[nodiscard]] constexpr auto reduction_op_for(R&&) -> typename C::reduction_op_type;
```

API Reference

Low-level MPI bindings

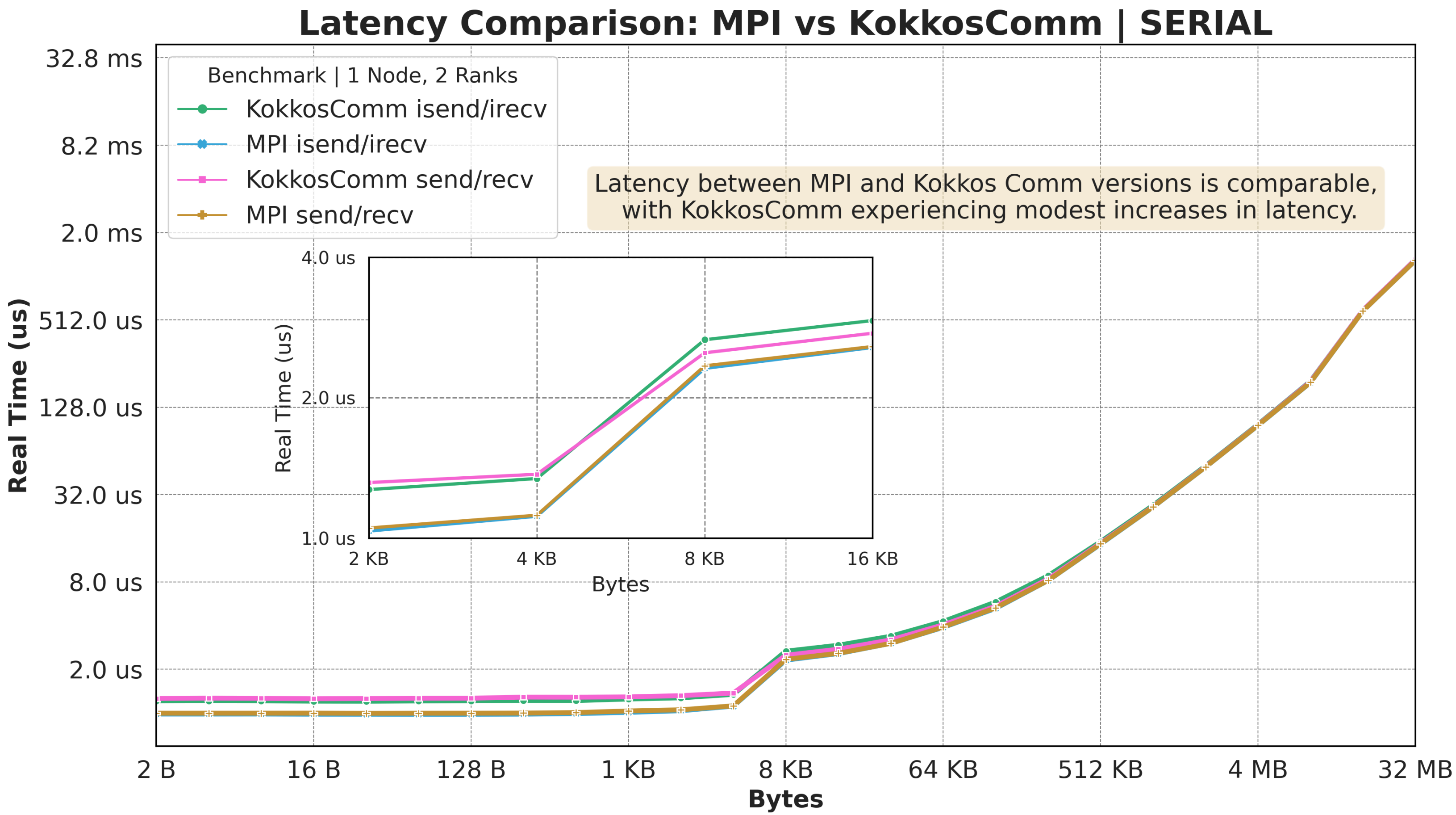
```cpp
template <Kokkos::ExecutionSpace E, KokkosComm::KokkosView V, KokkosComm::SendMode M>
auto KokkosComm::mpi::send(const E& exec, const V& view, int dst, MPI_Comm comm, [[maybe_unused]] M send_mode) -> void;
template <Kokkos::ExecutionSpace E, KokkosComm::KokkosView V, KokkosComm::SendMode M>
auto KokkosComm::mpi::isend(const E& exec, const V& view, int dst, MPI_Comm comm, [[maybe_unused]] M send_mode) -> KokkosComm::Request<MpiSpace>;

template <Kokkos::ExecutionSpace E, KokkosComm::KokkosView V>
auto KokkosComm::mpi::recv(const E& exec, const V& view, int src, MPI_Comm comm) -> void;
template <Kokkos::ExecutionSpace E, KokkosComm::KokkosView V>
auto KokkosComm::mpi::irecv(const E& exec, const V& view, int src, MPI_Comm comm) -> KokkosComm::Request<MpiSpace>;

template <Kokkos::ExecutionSpace E, KokkosComm::KokkosView V>
auto KokkosComm::mpi::broadcast(const E& exec, V& view, int root, MPI_Comm comm) -> void;
template <Kokkos::ExecutionSpace E, KokkosComm::KokkosView V>
auto KokkosComm::mpi::ibroadcast(const E& exec, V& view, int root, MPI_Comm comm) -> KokkosComm::Request<MpiSpace>;

template <Kokkos::ExecutionSpace E, KokkosComm::KokkosView SV, KokkosComm::KokkosView RV>
auto KokkosComm::mpi::all_gather(const E& exec, const SV& send_view, RV& recv_view, MPI_Comm comm) -> void;
template <Kokkos::ExecutionSpace E, KokkosComm::KokkosView SV, KokkosComm::KokkosView RV>
auto KokkosComm::mpi::iall_gather(const E& exec, const SV& send_view, RV& recv_view, MPI_Comm comm) -> KokkosComm::Request<MpiSpace>;

template <Kokkos::ExecutionSpace E, KokkosComm::KokkosView SV, KokkosComm::KokkosView RV>
auto KokkosComm::mpi::all_to_all(const E& exec, const SV& send_view, RV& recv_view, int count, MPI_Comm comm) -> void;
template <Kokkos::ExecutionSpace E, KokkosComm::KokkosView SV, KokkosComm::KokkosView RV>
auto KokkosComm::mpi::iall_to_all(const E& exec, const SV& send_view, RV& recv_view, int count, MPI_Comm comm) -> KokkosComm::Request<MpiSpace>;

template <Kokkos::ExecutionSpace E, KokkosComm::KokkosView SV, KokkosComm::KokkosView RV>
auto KokkosComm::mpi::all_reduce(const E& exec, const SV& send_view, RV& recv_view, MPI_Op red_op, MPI_Comm comm) -> void;
template <Kokkos::ExecutionSpace E, KokkosComm::KokkosView SV, KokkosComm::KokkosView RV>
auto KokkosComm::mpi::iall_reduce(const E& exec, const SV& send_view, RV& recv_view, MPI_Op red_op, MPI_Comm comm) -> KokkosComm::Request<MpiSpace>;

template <Kokkos::ExecutionSpace E, KokkosComm::KokkosView SV, KokkosComm::KokkosView RV>
auto KokkosComm::mpi::reduce(const E& exec, const SV& send_view, RV& recv_view, MPI_Op red_op, int root, MPI_Comm comm) -> void;
template <Kokkos::ExecutionSpace E, KokkosComm::KokkosView SV, KokkosComm::KokkosView RV>
auto KokkosComm::mpi::ireduce(const E& exec, const SV& send_view, RV& recv_view, MPI_Op red_op, int root, MPI_Comm comm) -> KokkosComm::Request<MpiSpace>;

template <Kokkos::ExecutionSpace E, KokkosComm::KokkosView SV, KokkosComm::KokkosView RV>
auto KokkosComm::mpi::inclusive_scan(const E& exec, const SV& send_view, RV& recv_view, MPI_Op red_op, MPI_Comm comm) -> void;

template <Kokkos::ExecutionSpace E, KokkosComm::KokkosView SV, KokkosComm::KokkosView RV>
auto KokkosComm::mpi::exclusive_scan(const E& exec, const SV& send_view, RV& recv_view, MPI_Op red_op, MPI_Comm comm) -> void;

explicit KokkosComm::mpi::Channel(int dst, int src, int tag, MPI_Comm comm);

template <KokkosComm::KokkosView V>
auto KokkosComm::mpi::Channel::sendinit(const V& view) -> void;

template <KokkosComm::KokkosView V>
auto KokkosComm::mpi::Channel::recvinit(V& view) -> void;

auto KokkosComm::mpi::Channel::start() -> void;
```

API Reference

Low-level NCCL bindings

```cpp
template <KokkosComm::KokkosView V>
auto KokkosComm::nccl::send(const Kokkos::Cuda& exec, const V& view, int dst, ncclComm_t comm) -> KokkosComm::Request<NcclSpace>;

template <KokkosComm::KokkosView V>
auto KokkosComm::nccl::recv(const Kokkos::Cuda& exec, const V& view, int src, ncclComm_t comm) -> KokkosComm::Request<NcclSpace>;

template <KokkosComm::KokkosView V>
auto KokkosComm::nccl::broadcast(const Kokkos::Cuda& exec, V& view, int root, ncclComm_t comm) -> KokkosComm::Request<NcclSpace>;

template <KokkosComm::KokkosView SV, KokkosComm::KokkosView RV>
auto KokkosComm::nccl::all_gather(const Kokkos::Cuda& exec, const SV& send_view, RV& recv_view, ncclComm_t comm) -> KokkosComm::Request<NcclSpace>;

template <KokkosComm::KokkosView SV, KokkosComm::KokkosView RV>
auto KokkosComm::nccl::all_to_all(const Kokkos::Cuda& exec, const SV& send_view, RV& recv_view, int count, ncclComm_t comm) -> KokkosComm::Request<NcclSpace>;

template <KokkosComm::KokkosView SV, KokkosComm::KokkosView RV>
auto KokkosComm::nccl::all_reduce(const Kokkos::Cuda& exec, const SV& send_view, RV& recv_view, ncclRedOp_t red_op, ncclComm_t comm) ->
KokkosComm::Request<NcclSpace>;

template <KokkosComm::KokkosView SV, KokkosComm::KokkosView RV>
auto KokkosComm::nccl::reduce(const Kokkos::Cuda& exec, const SV& send_view, RV& recv_view, ncclRedOp_t red_op, int root, int rank, ncclComm_t comm) ->
KokkosComm::Request<NcclSpace>;
```

Overhead: Kokkos Comm vs. raw MPI latency



Kokkos Comm infrastructure

- Build system
  - Modern CMake (3.25+)
  - Library packaging
  - *Workflow presets* for developement (in the works)
- CI/CD
  - PRs tested on comprehensive software stacks:
    - OS: Linux, macOS
    - Kokkos backends: OpenMP, Threads, CUDA, HIP
    - Communication libraries: Open MPI, MPICH, NCCL
  - Automatic checks: formating, linting, typos, etc.