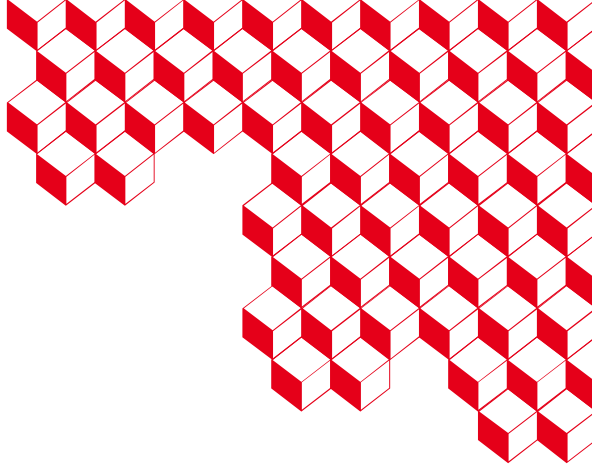


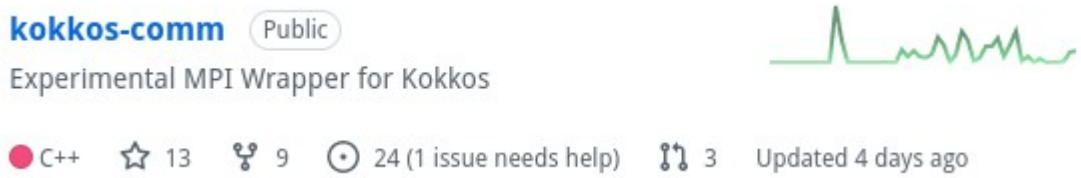
KokkosComm

A Communication Layer for Distributed Kokkos Applications

EuroMPI 2024, September 25-27 , Perth, Australia

Gabriel Dos Santos[1 2], C. 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
[2] Université Paris-Saclay, LIHPC, France
[3] Sandia National Laboratories, Albuquerque, NM, USA
[4] Tennessee Technological University, Cookeville, TN, USA



The Kokkos C++ Performance Portability Programming ecosystem

- Kokkos Core implements a programming model in C++ for writing performance portable applications targeting all major HPC platforms
- Kokkos is a basic building block for various applications
  - Trilinos
  - PETSc
  - LAMMPS
  - Cabana
  - ...
- Kokkos is OpenSource
- Kokkos is a High Performance Sofware Foundation project

Members

Premier

aws
Hewlett Packard Enterprise
Lawrence Livermore National Laboratory
Sandia National Laboratories

General

AMD
intel
NVIDIA
Argonne NATIONAL LABORATORY
OAK RIDGE National Laboratory
Los Alamos NATIONAL LABORATORY
kitware
cea

Associate

CDAC
UNIVERSITY OF OREGON
UNIVERSITY OF MARYLAND

The Kokkos C++ Performance Portability Programming ecosystem

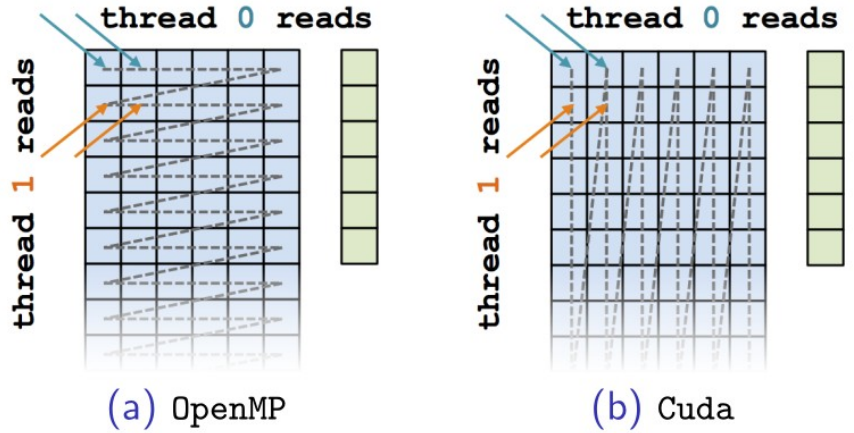
How it works?

- Simple parallel primitives:
  - `for`
  - `reduce`
  - `scan`

- Multi-dimensional data structure: Views
  - Data representation changes to optimize memory access performance on the target hardware
    - *LayoutRight* (row-major) on CPU
    - *LayoutLeft* (col-major) on GPU
  - Can express non-contiguous slices of data: subviews

```
View<double**, LayoutRight> A(N, M);
auto first_col_of_A = subview(A, Kokkos::ALL, 0);
assert(not first_col_of_A.span_is_contiguous());
```

```
View<double**, ExecutionSpace> A(N, M);
parallel_for(RangePolicy< ExecutionSpace>(0, N),
  ... thisRowsSum += A(j, i) * x(i);
```



- **HostSpace**: cached (good)
- **CudaSpace**: coalesced (good)

From Kokkos' tutorial

Extending Kokkos to distributed computing

No primitives for distributed computing

- HPC applications based on Kokkos must rely on external frameworks for distributed computing
  - Generally message-passing based: MPI, NCCL, RCCL, etc.
  - Also an official remote memory access library: kokkos-remote-spaces

- Challenges for programmers
  - Must handle implementation-defined specificities
    - Is the MPI GPU-aware?
    - Explicitly copy Views from CPU ⟺ GPU?
  - Must handle non-contiguous Views
    - Send as multiple "small" contiguous chunks
    - Pack/unpack as one "big" contiguous chunk

Extending Kokkos to distributed computing

No primitives for distributed computing

- HPC applications based on Kokkos must rely on external frameworks for distributed computing
  - Generally message-passing based: MPI, NCCL, RCCL, etc.
  - Also an official remote memory access library: kokkos-remote-spaces

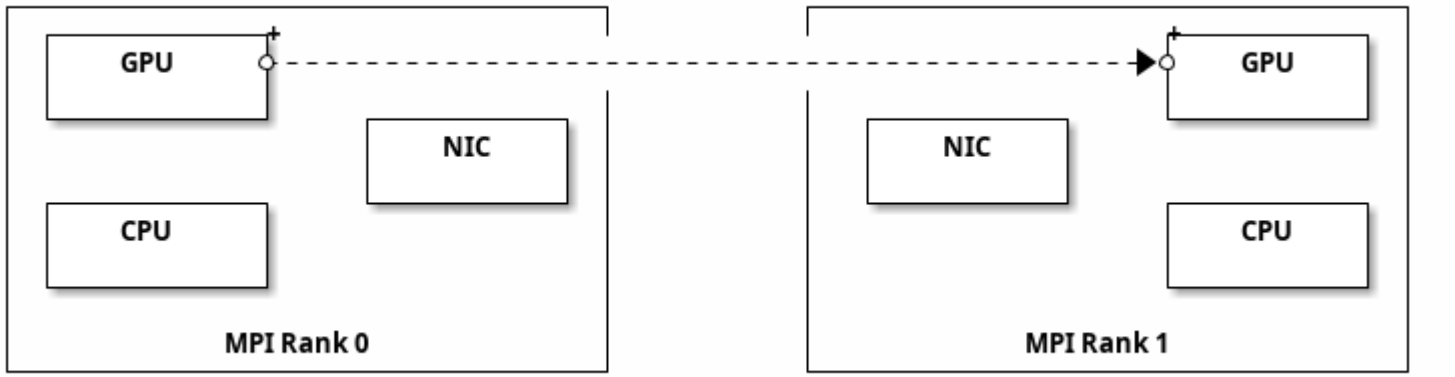
- Challenges for programmers
  - Must handle implementation-defined specificities
    - Is the MPI GPU-aware?
    - Explicitly copy Views from CPU ⟺ GPU?
  - Must handle non-contiguous Views
    - Send as multiple "small" contiguous chunks
    - Pack/unpack as one "big" contiguous chunk

⟹ Lots of code duplication across Kokkos projects

Motivations

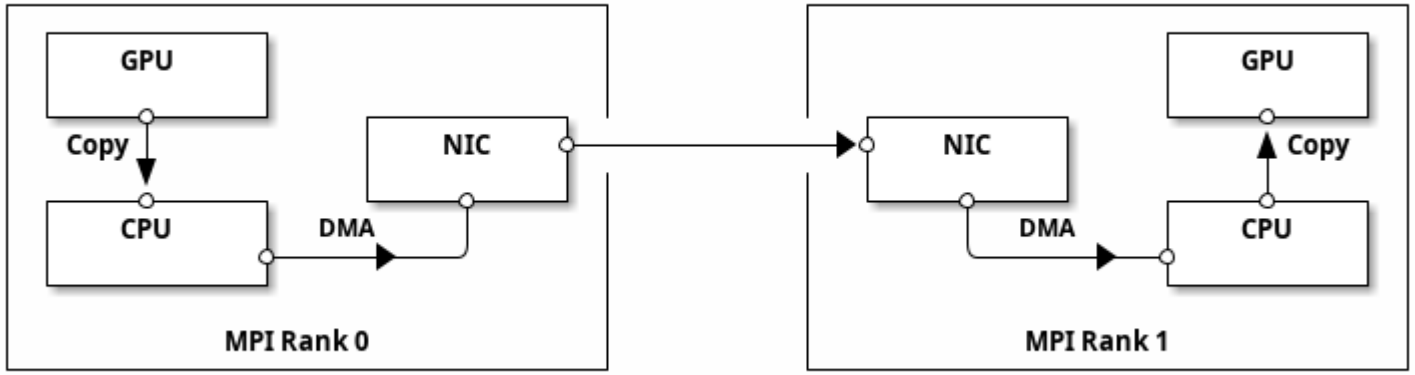
How do we communicate a Kokkos::View between the two MPI processes?

- If MPI is not GPU-aware
- If MPI is GPU aware



Motivations (non) GPU awareness

Message have to be copied on the CPU memory.

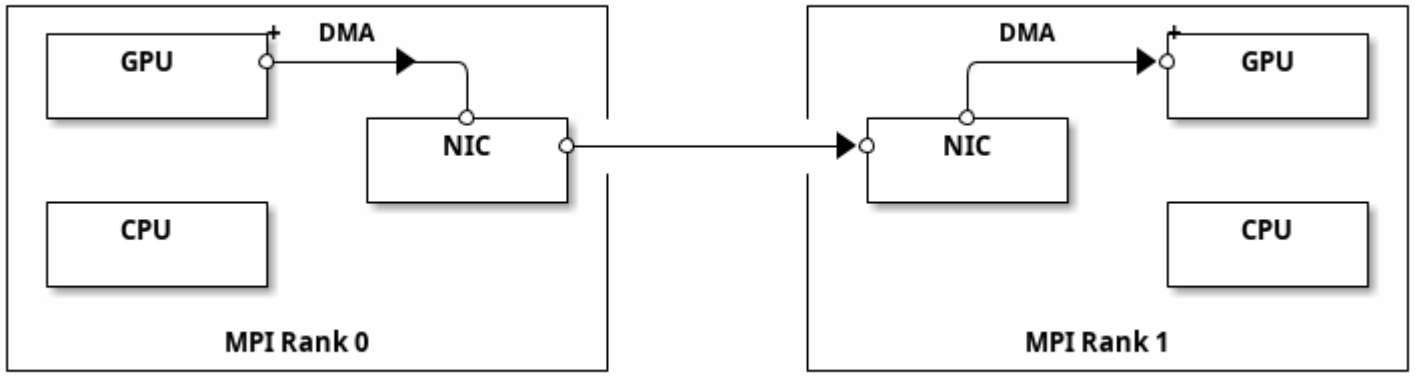



```
// Kokkos::View<double*, Kokkos::CudaSpace>
↪  to_send;
auto host_buffer =
  create_mirror_view_and_copy(to_send);
MPI_Send(host_buffer.span(),
  host_buffer.extent(0),
  MPI_DOUBLE, 1, TAG, MPI_COMM_WORLD);
```

```
// Kokkos::View<double*, Kokkos::CudaSpace>
↪  to_recv;
auto host_buffer =
  create_mirror_view(to_recv);
MPI_Recv(host_buffer.span(),
  host_buffer.extent(0),
  MPI_DOUBLE, 0, TAG, MPI_COMM_WORLD);
// Copy back to GPU
deep_copy(to_recv, host_buffer);
```

Motivations GPU-awareness

Message does not need to be copied on CPU memory!




```
// Kokkos::View<double*, Kokkos::CudaSpace>
↪  to_send;
MPI_Send(to_send.span(), to_send.extent(0),
  MPI_DOUBLE, 1, TAG, MPI_COMM_WORLD);
```

```
// Kokkos::View<double*, Kokkos::CudaSpace>
↪  to_recv;
MPI_Recv(to_recv.span(), to_recv.extent(0),
  MPI_DOUBLE, 0, TAG, MPI_COMM_WORLD);
```

Motivation : Kokkos specifics

Kokkos::Views are multi-dimentional arrays that can be stored in various ways:

- LayoutRight, LayoutLeft, LayoutStride, etc.
- Contiguous or non-contiguous (sub-views?)

Contiguous vs non-contiguous

```
if constexpr (to_send.span_is_contiguous()) {
  MPI_Send(to_send.span(), to_send.extent(0), MPI_DOUBLE,
           1, TAG, MPI_COMM_WORLD);
} else {
  Kokkos::View<...> send_buffer;
  Kokkos::deep_copy(send_buffer, to_send);
  MPI_Send(send_buffer.span(), send_buffer.extent(0), MPI_DOUBLE,
           1, TAG, MPI_COMM_WORLD);
}
```

- Kokkos execution contexts: interactions with asynchronous execution
  - When to fence() and where?

Related work

Kokkos + MPI interoperability interfaces

- Teuchos: MPI toolkit from Trilinos
  - MPI abstraction but no Kokkos::View abstraction

- Custom implementation in user applications
  - Cabana
  - LAMMPS
  - ...

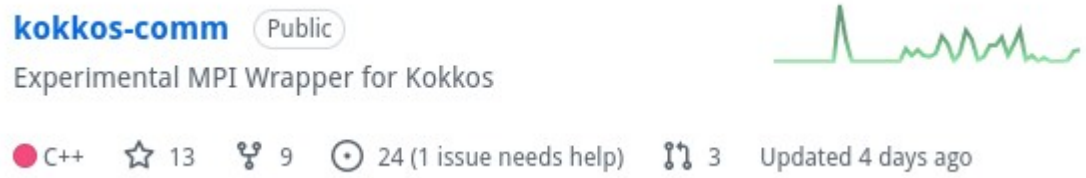
- ExaMPI: View-aware Message Passing
  - C++ implementation of MPI
  - Extends MPI API to deal with Kokkos Views
    - Still C-like interface: exposes counts, datatypes, etc.
    - Non-standard implementation that is harder to integrate into existing projects

Introducing KokkosComm

An official Kokkos organization project

- Collaborative development from CEA, SNL, Tenessee Technical University, etc.
- Provide a unified interface for distributed computing with Kokkos
- Launched on March 2024 !

Project goals (short term)



- Primarily address Kokkos + MPI interoperability
  - Simplify usage of both frameworks
  - Handle error-prone cases: non-contiguous data, GPU-aware support, etc.
  - Built on top of classical MPI implementation
- Guarantee performance portability across heterogeneous hardware
  - Automatically choose the best strategy for exchanging Views
- Make it hard to misuse
  - Simplify the API (so cannot follow stricly all the MPI semantics)
  - Leverage modern C++ features to make compile-time guarantees

Introducing KokkosComm

Project goals ("longer" term)

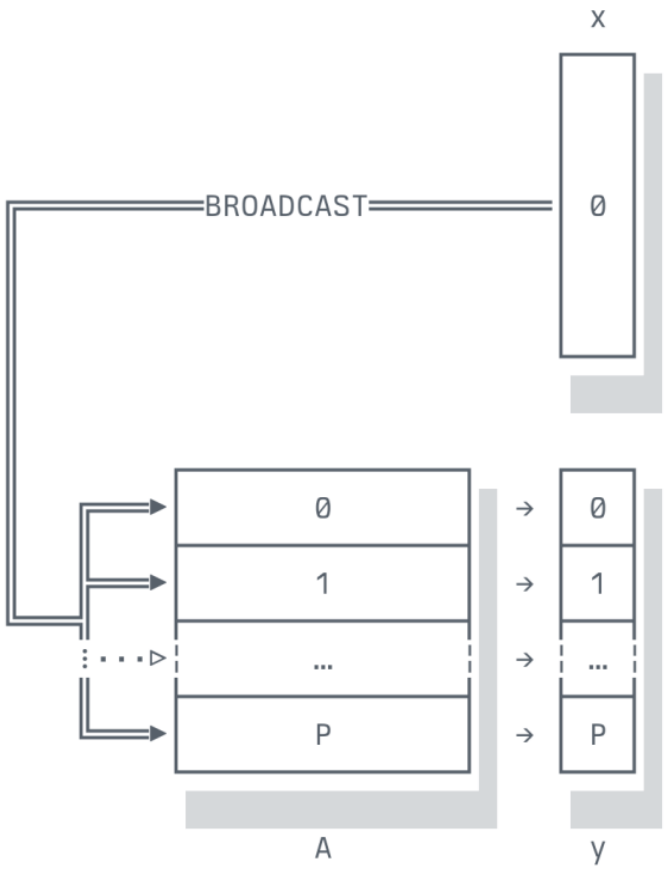
- Design a high-level, extensible and composable API
  - Abstract away concepts from underlying message-passing libraries
    - MPI, NCCL/RCCL, Portals, etc.
    - Communicators, requests, etc.
  - Provide a sensible subset of P2P & collective operations
    - Only non-blocking/asynchronous functions
    - P2P: **`send`** and **`recv`**
    - Collectives: **`broadcast`**, **`reduce`**, **`all_reduce`**, **`all_gather`** and **`all_to_all`**

- Overlap communication and data transformation
  - on-the-fly layout transformation of multi-dimensional data
  - e.g. transpose a 2D view in the correct layout (depending on comm target)

- Explore modern approaches for a new MPI API for C++
  - Deal with multi-dimensional data with `std::mdspan` (C++23)
  - Express dependencies between operations with `std::execution`'s sender/receiver model (C++26)

Distributed matrix-vector product (1D row-wise partitioning)

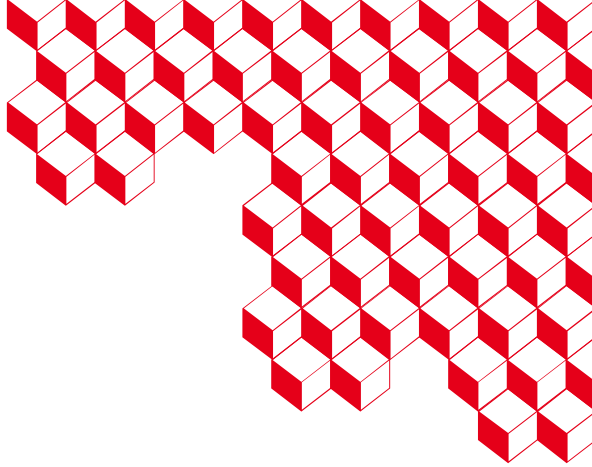
```cpp
// Initialize vector `x` from the root process, and broadcast it
Vector x("x", global_n);
if (rank == 0) {
  Kokkos::parallel_for("init_x", global_n, KOKKOS_LAMBDA(int i) { x(i) = 3.0; });
}
auto bcast_req = KokkosComm::broadcast(space, handle, x, 0);
// Local Matrix `A` and result vector `y`
Matrix A_local("local_A", local_m, global_n);
Vector y_local("local_y", local_m);
// Initialize local `A` on each process
Kokkos::parallel_for("init_local_A",
  Kokkos::MDRangePolicy<Kokkos::Rank<2>>({0, 0}, {local_m, global_n}),
  KOKKOS_LAMBDA(int i, int j) { A_local(i, j) = 1.0; }
);
// Wait for broadcast of `x` to finish
KokkosComm::wait(bcast_req);

// Perform local matrix-vector multiplication
KokkosBlas::gemv("N", alpha, A_local, x, beta, y_local);

// Gather results (if needed)
Vector y("y", global_m);
auto gather_req = KokkosComm::all_gather(space, handle, y_local, y);
KokkosComm::wait(gather_req);
```



Summary & Take away

- Basic MPI wrapper for Kokkos::Views by the end of the year
  - To be tested on Trilinos and PETSc
  - Looking for different communication patterns (LAMMPS, Cabana, your app?)

- Open development
  - https://github.com/kokkos/kokkos-comm
  - #mpi-interop on Kokkos' Slack
  - Bi-weekly telecom, open to all (you are welcome!)



Thank you for your attention!

Questions & Answers