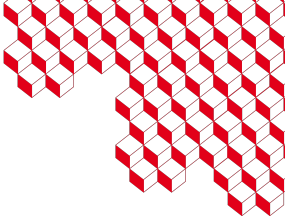


Kokkos MPI Interop

CExA Tea-Time 2024-06-19

C. Chevalier

Outline

Context

`Kokkos` targets one node and traditionnaly one accelerator

Problem

How to scale up an application to exploit:

- several accelerators on the same node
- several nodes

Solution

- Use `Kokkos Remote Spaces`
- Use `MPI`
  - by hand
  - using a shared wrapper `KokkosComm`

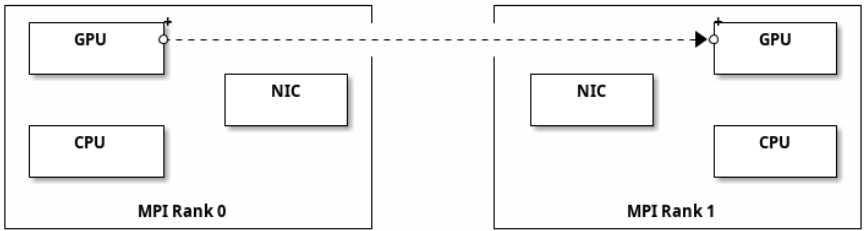
Goals

- Provide a zero-cost wrapper on top of `MPI` to communicate `Kokkos::View`
  - Works on a standard `MPI` implementation
  - Deals with GPU/cuda/hip awareness
- Interop with `MPI` calls of legacy parts of the application
- Explore newer features:
  - Asynchronism
  - Device-initiated communications
  - Network offloading (smart NICs)
- Proposal of a `mdspan` API in `MPI` standard

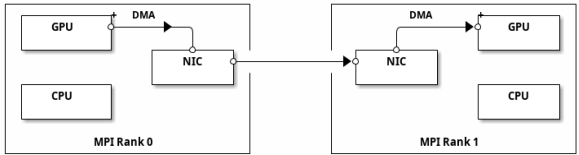
Motivations

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

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



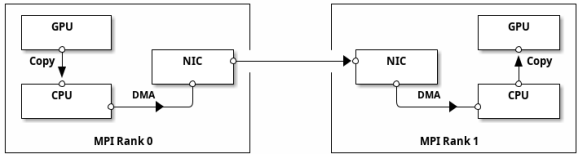
Motivation, GPU-awareness




```
// 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, (non) GPU awareness




```cpp
// 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);
```

```cpp
// 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);
```

Motivation: Kokkos specificities

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

- `LayoutRight`, `LayoutLeft`, `LayoutStride`, ...
- contiguous or non-contiguous (subviews?)

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?

Outline

KokkosComm

- Very new project in `Kokkos` github organization `https://github.com/kokkos/kokkos-comm`
- Name is not final yet!
- Launched in March 2024, from slack #mpi-interop
- Direct involvement of
  - CEA
  - Sandia
  - Tennessee Tech
  - More to come (CERFACS, ORNL, …)

Current work

- Bootstrapping a collaborative project inspired by `kokkos` workflow
  - PRs and reviews
  - Currently 2 maintainers for the project (CEA and Sandia)
- Implementing basic wrapper for 1D `View` and multi-d contiguous views
  - similar to `Trilinos`' `Teuchos` MPI wrapper
  - with a more modern C++ API
- Design discussions on:
  - Error handling
  - Interactions with `Kokkos::Space`
  - Extensions for multi-dimensional arrays
  - ...

Small Example

```
template <KokkosComm::CommMode SendMode, typename Scalar>
void send_comm_mode_1d_noncontig() {

  // this is C-style (row) layout, i.e. b(0,0) is next to b(0,1)
  Kokkos::View<Scalar **, Kokkos::LayoutRight> b("b", 10, 10);
  auto a = Kokkos::subview(b, Kokkos::ALL, 2);  // take column 2 (non-contiguous)

  int rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  if (0 == rank) {
    int dst = 1;
    Kokkos::parallel_for(
        a.extent(0), KOKKOS_LAMBDA(const int i) { a(i) = i; });
    KokkosComm::send<SendMode>(Kokkos::DefaultExecutionSpace(), a, dst, 0, MPI_COMM_WORLD);
  } else if (1 == rank) {
    int src = 0;
    KokkosComm::recv(Kokkos::DefaultExecutionSpace(), a, src, 0, MPI_COMM_WORLD);
  }
}
```

Perspectives

- Simple API
  - Only a subset of MPI functions
  - Avoid too many arguments, sometimes unused in MPI
- Performance optimization
  - Use of `MPI Session` to fit our needs (for example, avoiding location pointer checking if we know it is a `View` on a GPU)
  - Multi-NICs
- Full asynchronous support:
  - `Sender/Receiver` C++ proposal P2300
- Device initiated communications
  - Using `NCCL` (or `*CCL`)
- Smart NICs (e.g., DPU usage) or custom network

Device initiated communications

This pattern should be possible from host or device:

```
using member_type = ExecutionSpace::member_type;
auto policy = Kokkos::TeamPolicy<>(comm_context.exec_space(), league_size, Kokkos::AUTO());

Kokkos::parallel_for(policy, KOKKOS_LAMBDA (member_type team_member) {
    [ ... ]
    // Send a subview to a remote process
    // Syntax to be determined
    auto req = KokkosComm::send(comm_context, send_view, destination);
    [ ... ]
});
```

Open question: how?

- With `OpenMP`, `MPI` with `MPI_THREAD_MULTIPLE`
- On device , `*CCL*`, but we have to keep the same semantic: blocking? synchronous?

Summary

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 application?)

Open development

- `https://github.com/kokkos/kokkos-comm`
- #mpi-interop on `Kokkos`' slack
- bi-weekly telecon, open to all