The Kokkos Lectures

Module 9: Kokkos-Fortran-Interop

May 20, 2026

Sandia National Laboratories is a multi-mission 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-NA0003525.
SAND2020-9315 TR

Online Resources:

- https://github.com/kokkos:
  - Primary Kokkos GitHub Organization
- https://github.com/kokkos/kokkos-tutorials/wiki/Kokkos-Lecture-Series:
  - Slides, recording and Q&A for the Lectures
- https://kokkos.github.io/kokkos-core-wiki:
  - Wiki including API reference
- https://kokkosteam.slack.com:
  - Slack channel for Kokkos.
  - Please join: fastest way to get your questions answered.
  - Can whitelist domains, or invite individual people.

- Repository:
  https://github.com/kokkos/kokkos-fortran-interop
- Requirements:
  - Kokkos 4.0 or newer
  - C++17/Fortran08 compiler suites
- Configure

```
cmake -DKokkos_ROOT=/kokkos/path /interop/path
```

- Kokkos-Fortran offers wrappers around:
  - `Kokkos::initialize(argc, argv)`
  - `Kokkos::finalize()`
  - `Kokkos::print_configuration(output)`
  - `Kokkos::View`
  - `Kokkos::DualView`
- User kernels are written in C++ $\rightarrow$ need to use `iso_c_binding`
- Only a subset of Kokkos capabilities are exposed

- Similar to MPI, Kokkos needs to be initialized by calling `kokkos_initialize` and finalized by calling `kokkos_finalize`
- The `kokkos_initialize` subroutine initializes Kokkos and reads command line arguments. It should be called after `MPI_Initialize`
- `kokkos_print_configure("output.txt")` prints the Kokkos configuration to the file `output.txt`

```
program my_kokkos_code
  use :: flcl_util_kokkos_mod

  ! Initialize Kokkos
  ! This subroutine reads command line arguments
  call kokkos_initialize()

  ! Print the configuration in a file
  call kokkos_print_configure('kokkos.out')

  ! Finalize Kokkos
  call kokkos_finalize()
end program my_kokkos_code
```

- `Kokkos::View` is Kokkos equivalent to an array
- `Kokkos::View` can have up to 8 dimensions in C++ but they are limited to 7 dimensions in Fortran due to limitation of the library
- Supported Fortran types: logical, 32-bit integer, 64-bit integer, 32-bit real, 64-bit real, 32-bit complex, 64-bit complex, and index (positive 64-bit integer)
- Types follow the pattern `view_<type>_<dimension>_t`, e.g., `view_r64_1d_t` is a one-dimensional view of 64-bit real

- The memory space defines where the memory is allocated
- Supported memory spaces: `Kokkos::HostSpace`, `Kokkos::CudaManagedSpace`, and `Kokkos::HIPManagedSpace`
- The memory space is determined during the configuration of the Kokkos-Fortran-Interop library, based on the memory space configuration of Kokkos.

- `Kokkos::View` can be allocated directly or built from an array. In the latter case, the `Kokkos::View` can only be used on the host
- `Kokkos::View` that are allocated with `kokkos_allocate_view` must also be deallocated with `kokkos_deallocate_view`
- `kokkos_allocate_view` initializes all the elements of the `Kokkos::View` to zero
- `Kokkos::View` cannot be accessed directly from Fortran instead it can be accessed through a `pointer`

```
use, intrinsic :: iso_c_binding
use, intrinsic :: iso_fortran_env
use :: flcl_mod

! Kokkos View only accessible from C++
type(view_r64_1d_t) :: v_c_y
! Pointer to access the Kokkos View from Fortran
real(real64), pointer, dimension(:) :: c_y
integer :: mm = 5000

call kokkos_allocate_view(c_y, v_c_y, 'c_y', &
                          int(mm, c_size_t))

! Do stuff

call kokkos_deallocate_view(c_y, v_c_y)
```

- `Kokkos::DualView` are similar to `Kokkos::View` but they are composed of two `Kokkos::Views`, one on the host and one on the device.
- It is the user's responsibility to synchronize the data
- The synchronization must be done in C++
- Supported memory spaces: `Kokkos::HostSpace`, `Kokkos::Cuda`, `Kokkos::HIP`, and `Kokkos::SYCL`
- `Kokkos::DualView` cannot be accessed directly from Fortran instead it can be accessed through a `pointer`

```
use, intrinsic :: iso_c_binding
use, intrinsic :: iso_fortran_env
use :: flcl_mod

real(real64), pointer, dimension(:) :: c_y
type(dualview_r64_1d_t) :: v_c_y
integer :: mm = 5000

call kokkos_allocate_dualview(c_y, v_c_y, 'c_y', &
                              int(mm, c_size_t))

! Do stuff

call kokkos_deallocate_dualview(c_y, v_c_y)
```

- Kernels using Kokkos are written in C++
- Use C-binding from Fortran standard
- Create a subroutine that calls the C++ function

```
use, intrinsic :: iso_c_binding
use, intrinsic :: iso_fortran_env
use :: flcl_mod
use :: my_init_mod

real(real64), pointer, dimension(:) :: c_y
type(view_r64_1d_t) :: v_c_y
integer :: mm = 5000

call kokkos_allocate_view(c_y, v_c_y, 'c_y', &
                          int(mm, c_size_t))

call my_init(v_c_y)

call kokkos_deallocate_view(c_y, v_c_y)
```

```
module my_init_mod
    use, intrinsic :: iso_c_binding
    use, intrinsic :: iso_fortran_env
    use :: flcl_mod
    implicit none
    public
      interface
        subroutine my_f_init( y ) &
          & bind(c, name='my_c_init')
          import
          type(c_ptr), intent(in) :: y
        end subroutine my_f_init
      end interface
```

```
    contains

      subroutine my_init( y )
        type(view_r64_1d_t), intent(inout) :: y
        call my_f_init(y%ptr())
      end subroutine my_init

end module my_init_mod
```

```cpp
#include <Kokkos_Core.hpp>
#include <flcl-cxx.hpp>
using view_type = flcl::view_r64_1d_t;

extern "C" {
  void my_c_init(view_type **v_y) {
    view_type y = **v_y;
    Kokkos::parallel_for(
        "init", y.extent(0),
        KOKKOS_LAMBDA(int idx) {y(idx) += idx;});
    Kokkos::fence();
  }
}
```

- Use `Kokkos::View` to do an *axpy*
- Do not forget to install the library

Fortran Language Compatibility Layer

- Initialize Kokkos from Fortran via `kokkos_initialize` and `kokkos_finalize`
- `nd_array_t` is a representation of a `Kokkos::View`
- Create `nd_array_t` from a Fortran array via `to_nd_array`
- Allocate `Kokkos::DualView` in Fortran with `kokkos_allocate_dualview`