LA-UR-18-26713
Approved for public release; distribution is unlimited.

Title: Tutorial Exercises Fortran Interface for Kokkos

Author(s): Womeldorff, Geoffrey Alan

Intended for: Kokkos Workshop @ ORNL, 2018-07-23/2018-07-27 (Oak Ridge, Tennessee, United States)

Issued: 2018-07-20

Tutorial Exercises

Fortran Interface for Kokkos

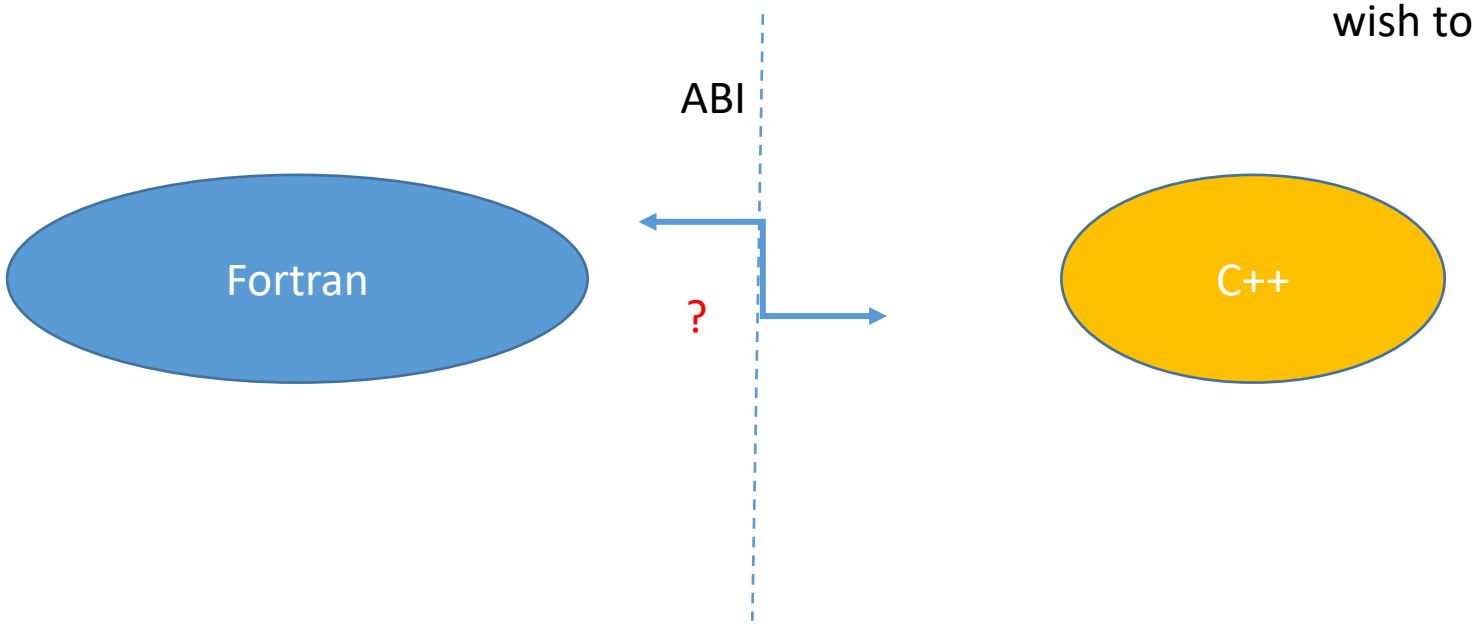
Geoff Womeldorff womeld@lanl.gov

Kokkos Tutorial @ ORNL

Fortran Interface for Kokkos

- <u>Problem:</u> Have production Fortran code, but want access to “exotic” memory and execution spaces. Want to adopt / translate incrementally.

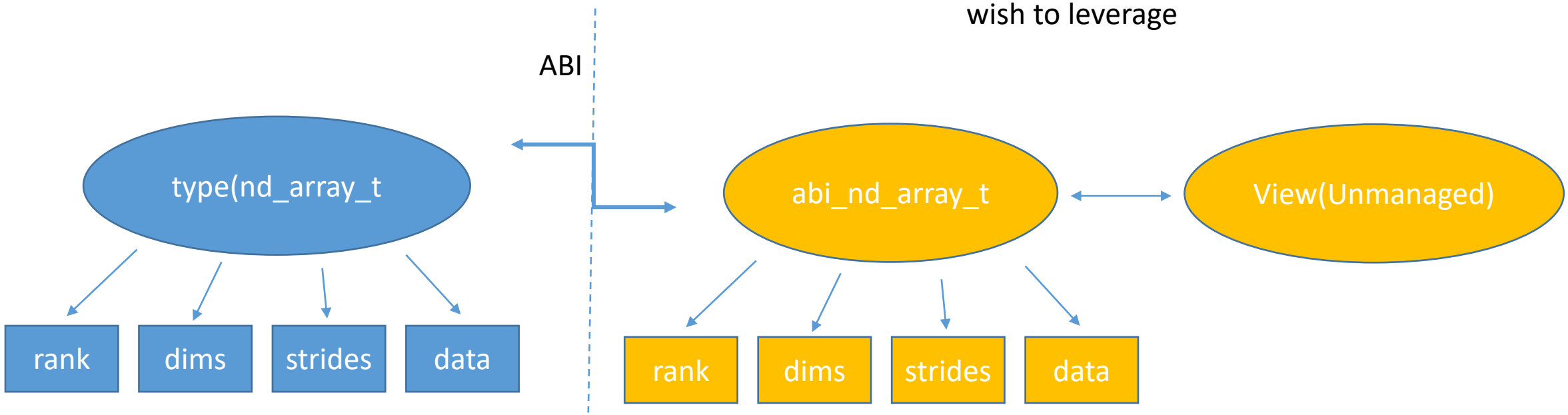
- Want a solution that will work across compilers and compiler families
- Based on and/or working towards standards
  - ISO_C_BINDING: F08 /F15
  - Mdarray C++17 / C++20
- May already have C++ kernels or libraries you wish to leverage



Fortran Interface for Kokkos

- Problem: Have production Fortran code, but want access to “exotic” memory and execution spaces. Want to adopt / translate incrementally.

- Want a solution that will work across compilers and compiler families
- Based on and/or working towards standards
  - ISO_C_BINDING: F08 /F15
  - Mdarray C++17 / C++20
- May already have C++ kernels or libraries you wish to leverage



Fortran Interface for Kokkos exercise

- Goal: couple Fortran memory allocations to C++ axpy kernel
  1. Instantiate an nd_array derived type for input arrays on Fortran side
  2. Create a View from nd_array struct on C++ side

Backup Slides

Source code for tutorial example

main.f90: 1 / 2

```
program main
use, intrinsic :: iso_c_binding
use, intrinsic :: iso_fortran_env

use :: abi_mod
use :: f_interface_mod

implicit none
integer :: n
real(c_double) :: alpha
real(c_double), dimension(:), allocatable :: array_x
real(c_double), dimension(:), allocatable :: f_array_y, c_array_y

n = 20

allocate( array_x(n) )
allocate( c_array_y(n) )
allocate( f_array_y(n) )
```

2 / 2

```
alpha = 0.5
array_x = 1
f_array_y = 1
c_array_y = 1

! f axpy
f_array_y = alpha * array_x + f_array_y

! alpha = 2.0

! c_axpy
call axpy_kokkos( alpha, array_x, c_array_y )

if ( abs(sum(f_array_y) - sum(c_array_y)) .le. 1.0**(-15) ) then
write(*,*)'Good job!'
else
write(*,*)'Please try again.'
end if

end program main
```

f_interface.f90: 1 / 4

```
module f_interface_mod

use, intrinsic :: iso_c_binding
use, intrinsic :: iso_fortran_env
use :: abi_mod
implicit none

public

interface
subroutine f_kokkos_initialize() &
bind(c, name='c_kokkos_initialize')
use, intrinsic :: iso_c_binding
implicit none
end subroutine f_kokkos_initialize
end interface
```

2 / 4

```
interface
subroutine f_kokkos_finalize() &
bind(c, name='c_kokkos_finalize')
use, intrinsic :: iso_c_binding
implicit none
end subroutine f_kokkos_finalize
end interface

interface
subroutine f_axpy_kokkos( alpha, nd_array_x, nd_array_y ) &
bind(c, name="c_axpy_kokkos")
use, intrinsic :: iso_c_binding
use :: abi_mod
implicit none
real(c_double), intent(inout) :: alpha
type(nd_array_t), intent(inout) :: nd_array_x
type(nd_array_t), intent(inout) :: nd_array_y
end subroutine f_axpy_kokkos
end interface
```

f_interface.f90: 3 / 4

```
contains

subroutine kokkos_initialize()
use, intrinsic :: iso_c_binding
implicit none
call f_kokkos_initialize()
end subroutine kokkos_initialize
subroutine kokkos_finalize()
use, intrinsic :: iso_c_binding
implicit none
call f_kokkos_finalize()
end subroutine kokkos_finalize

subroutine axpy_kokkos( alpha, array_x, array_y )
use, intrinsic :: iso_c_binding
use :: abi_mod
implicit none
real(c_double) :: alpha
real(c_double), dimension(:), intent(inout) :: array_x
real(c_double), dimension(:), intent(inout) :: array_y
```

4 / 4

```
type(nd_array_t) :: nd_array_x
type(nd_array_t) :: nd_array_y
integer(c_size_t), target :: array_x_dims(1)
integer(c_size_t), target :: array_y_dims(1)
integer(c_size_t), target :: array_x_stride(1)
integer(c_size_t), target :: array_y_stride(1)
nd_array_x = to_nd_array( array_x, array_x_dims, array_x_stride )
nd_array_y = to_nd_array( array_y, array_y_dims, array_y_stride )

call f_axpy_kokkos( alpha, nd_array_x, nd_array_y )
end subroutine axpy_kokkos

end module f_interface_mod
```

c_interface.cpp: 1 / 1

```
#include "abi.hpp"
extern "C" {

void c_kokkos_initialize() {  Kokkos::initialize();  }

void c_kokkos_finalize( void ) {  Kokkos::finalize();  }

void c_axpy_kokkos( double &alpha, abi_ndarray_t &nd_array_x,
                                   abi_ndarray_t &nd_array_y ) {

auto array_x = view_from_ndarray<double*>(nd_array_x);
auto array_y = view_from_ndarray<double*>(nd_array_y);

// y = alpha*x + y
Kokkos::parallel_for(nd_array_x.dims[0],
   KOKKOS_LAMBDA (const size_t ii) { array_y(ii) += alpha * array_x(ii); }
);

}

}
```

abi.f90: 1 / 2

```
module abi_mod
use, intrinsic :: iso_c_binding
use, intrinsic :: iso_fortran_env

implicit none
private

public nd_array_t
public to_nd_array

type, bind(C) :: nd_array_t
integer(c_size_t) :: rank
type(c_ptr) :: dims
type(c_ptr) :: strides
type(c_ptr) :: data
end type nd_array_t

interface to_nd_array
module procedure to_nd_array_r64_1d
end interface
contains
```

2 / 2

```
function to_nd_array_r64_1d(array, dims, strides) result(ndarray)
real(REAL64), target, intent(in) :: array(:)
integer(c_size_t), target, intent(inout) :: dims(1)
integer(c_size_t), target, intent(inout) :: strides(1)
type(nd_array_t) :: ndarray

dims(1) = size(array, 1, c_size_t)
if (size(array, 1) .ge. 2) then
strides(1) = &
(transfer(c_loc(array(2)), 1_c_size_t) - &
transfer(c_loc(array(1)), 1_c_size_t)) / c_sizeof(array(1))
else
strides(1) = 0
end if

ndarray%rank = 1
ndarray%dims = c_loc(dims(1))
ndarray%strides = c_loc(strides(1))
ndarray%data = c_loc(array(1))
end function to_nd_array_r64_1d

end module
```

abi.hpp: 1 / 3

```
#include <stddef.h>
#include <Kokkos_Core.hpp>

extern "C" {

typedef struct _abi_nd_array_t {
size_t rank;
size_t const *dims;
size_t const *strides;
void *data;
} abi_ndarray_t;

}
```

2 / 3

```
template <typename DataType>

Kokkos::View<DataType, Kokkos::LayoutStride, Kokkos::HostSpace,
Kokkos::MemoryUnmanaged>

view_from_ndarray(abi_ndarray_t const &ndarray) {

size_t dimensions[Kokkos::ARRAY_LAYOUT_MAX_RANK] = {};

size_t strides[Kokkos::ARRAY_LAYOUT_MAX_RANK] = {};


using traits = Kokkos::ViewTraits<DataType>;

using value_type = typename traits::value_type;

constexpr auto rank = Kokkos::ViewTraits<DataType>::rank;


if (rank != ndarray.rank) {

std::cerr << "Requested Kokkos view of rank " << rank << " for ndarray with rank
"

<< ndarray.rank << "." << std::endl;

std::exit(EXIT_FAILURE);

}
```

abi.hpp: 3 / 3

```
std::copy(ndarray.dims, ndarray.dims + ndarray.rank, dimensions);

std::copy(ndarray.strides, ndarray.strides + ndarray.rank, strides);


// clang-format off

Kokkos::LayoutStride layout{

dimensions[0], strides[0],

dimensions[1], strides[1],

dimensions[2], strides[2],

dimensions[3], strides[3],

dimensions[4], strides[4],

dimensions[5], strides[5],

dimensions[6], strides[6],

dimensions[7], strides[7]

};

// clang-format on


return Kokkos::View<DataType, Kokkos::LayoutStride, Kokkos::HostSpace,
Kokkos::MemoryUnmanaged>(

reinterpret_cast<value_type *>(ndarray.data), layout);

}
```

compile.sh: 1 / 1

```
#!/bin/bash
export KOKKOS_ROOT_DIR=/KOKKOS/DIR/HERE
rm *.o *.mod *.x
gfortran -c -std=f2008 abi.f90
gfortran -c -std=f2008 f_interface.f90
g++ -c -fopenmp -I. -I$KOKKOS_ROOT_DIR/include c_interface.cpp
gfortran -c -g -std=f2008 main.f90
gfortran -std=f2008 -o ftest.x abi.o f_interface.o c_interface.o main.o -L$KOKKOS_ROOT_DIR/lib -lkokkos -lstdc++ -fopenmp
```