MPI and Modern Fortran: Better Together

Jeff Hammond
Principal Architect
HPC Software

Abstract

This talk will describe the right way to use MPI in Fortran. The MPI Fortran 2008 bindings offer type safety and are the only standard-compliant method for using MPI in Fortran. I will show how and why to use MPI_F08. One challenge is imperfect implementations of MPI Fortran support. I will show how this is solved using Vapaa, a standalone implementation of MPI_F08. I will also show how to make Fortran code more efficient with wrappers that eliminate the need for redundanent arguments, i.e. count and type can be inferred from Fortran arrays. Finally, I will talk about the current activities of the MPI Forum as they relate to Fortran.

Outline

1. Why **mpi_f08** is awesome and you should use it.
2. Vapaa: implementing **mpi_f08** independent of MPI C libraries.
3. Havaita: improving usability of MPI using modern Fortran.
4. Summary of MPI-Next.

MPI Language Support

1. MPI C API - used by all languages except Fortran
2. ~~MPI C++ API~~ deleted in MPI 3.0 (2012)
3. MPI **mpif.h** (falsely known as “F77” bindings)
4. MPI **mpi** module (falsely known as “F90” bindings)
5. MPI **mpi_f08** module (the good stuff)

NVIDIA.

MPI Language Support

1. MPI C API - used by all languages except Fortran
2. ~~MPI C++ API~~ deleted in MPI 3.0 (2012)
3. ~~MPI mpif.h~~ deprecated in MPI 4.1 (2023)
4. MPI mpi module (falsely known as “F90” bindings)
5. MPI mpi_f08 module (the good stuff)

MPI Fortran legacy API

```
MPI_BCAST(BUFFER, COUNT, DATATYPE, ROOT, COMM, IERROR)
    !$PRAGMA IGNORE_TKR
    <type> :: BUFFER(*)
    INTEGER :: COUNT, DATATYPE, ROOT, COMM, IERROR
```

Until Fortran 2008, there is no standard mechanism for type-agnostic buffers equivalent to C void*. Implementations rely on compiler-specific extensions (e.g., as shown above) or lack of enforcement of type safety to compile. There is also no way to use Fortran array properties, including subarrays.

NVIDIA.

MPI Fortran legacy API

```
MPI_BCAST(BUFFER, COUNT, DATATYPE, ROOT, COMM, IERROR)
    !$PRAGMA IGNORE_TKR
    <type> :: BUFFER(*)
    INTEGER :: COUNT, DATATYPE, ROOT, COMM, IERROR
```

Datatypes and communicators are MPI object handles, not generic integers. Without proper types, compilers cannot identify user errors, so they manifest in unpleasant ways at runtime.

NVIDIA.

MPI Fortran modern API

```
MPI_Bcast(buffer, count, datatype, root, comm, ierror)
    TYPE(*), DIMENSION(..) :: buffer
    INTEGER, INTENT(IN) :: count, root
    TYPE(MPI_Datatype), INTENT(IN) :: datatype
    TYPE(MPI_Comm), INTENT(IN) :: comm
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror
```

Buffers are assumed-type, assumed-rank arguments. MPI implementations can - but are not required to - support non-contiguous subarrays.

MPI Fortran modern API

```
MPI_Bcast(buffer, count, datatype, root, comm, ierror)
    TYPE(*), DIMENSION(..) :: buffer
    INTEGER, INTENT(IN) :: count, root
    TYPE(MPI_Datatype), INTENT(IN) :: datatype
    TYPE(MPI_Comm), INTENT(IN) :: comm
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror
```

MPI object handles are properly typed and thus compilers will not accept erroneous usage. At the same time, MPI object handle types are interoperable with the old method, because the type contains the integer handle as its only member.

NVIDIA.

MPI Fortran modern API

```
MPI_Bcast(buffer, count, datatype, root, comm, ierror)
    TYPE(*), DIMENSION(..) :: buffer
    INTEGER(KIND=MPI_COUNT_KIND), INTENT(IN) :: count
    TYPE(MPI_Datatype), INTENT(IN) :: datatype
    INTEGER, INTENT(IN) :: root
    TYPE(MPI_Comm), INTENT(IN) :: comm
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror
```

All MPI procedures that take a count argument use polymorphic interfaces to support both INTEGER (usually 32b) and large-count (i.e. 64b) variants. This aspect of the MPI Fortran API is superior to both C and C++. (C11 _Generic was rejected and C++ polymorphism can't exist without native bindings.)

NVIDIA.

MPI Fortran modern API

```
MPI_Irecv(buf, count, datatype, source, tag, comm,
request, ierror)
    TYPE(*), DIMENSION(..), ASYNCHRONOUS :: buf
    INTEGER, INTENT(IN) :: count, source, tag
    TYPE(MPI_Datatype), INTENT(IN) :: datatype
    TYPE(MPI_Comm), INTENT(IN) :: comm
    TYPE(MPI_Request), INTENT(OUT) :: request
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror
```

Non-blocking procedures require ASYNCHRONOUS buffer attribute to prohibit (unlikely) compiler optimizations and (likely) temporary copies that break correctness.

NVIDIA.

Asynchronous Procedures

MPI_SUBARRAYS_SUPPORTED = .FALSE.

MPI_ASYNC_PROTECTS_NONBLOCKING = .FALSE.

```
integer :: buf(1000)
call MPI_Irecv(buf(1:1000:2), .., req)
    ! integer :: temp(500) = buf(1:1000:2)
    ! address of temp passed to network API
    ! temp is deallocated when MPI_Irecv returns
call MPI_Wait(req)
    ! network writes to temp, which no longer exists
    ! segmentation fault
```

Asynchronous Procedures

MPI_SUBARRAYS_SUPPORTED = .TRUE.

MPI_ASYNC_PROTECTS_NONBLOCKING = .TRUE.

```
void CFI_MPI_Irecv(CFI_cdesc_t * desc, int count, int datatype_f, int
source, int tag, int comm_f, int * request_f, int * ierror)
{
    ! translate datatype and communication handles from Fortran to C
    ! create MPI datatype from desc, count and datatype: temp_type
    *ierror = MPI_Irecv(desc->base_addr, 1, temp_type, source, tag,
comm, &request);
    ! translate request handle from C to Fortran
}
```

https://github.com/jeffhammond/vapaa/blob/main/source/mpi_p2p.c

NVIDIA

Implementation Status

Today:

- Fortran compiler must support Fortran 2018 CFI features, i.e. CFD_cdesc_t, or non-contiguous subarrays not supported efficiently and correctly.
- MPI implementation must support assumed-rank, assumed-type arguments in the interface and CFI_cdesc_t buffer arguments in the implementation.
- MPI Fortran features must be compiled along with the entire MPI implementation, once for every Fortran compiler. MPI library build time dominated by C object files.

Tomorrow:

- Fortran compiler should support Fortran 2018 CFI features, i.e. CFD_cdesc_t, but compiler-specific descriptions can be used instead.
- MPI Fortran features provided by third-party library that depends only on the C API or ABI.
- MPI implementation only used for Fortran handle conversion, f2c+c2f, optionally.

VAPAA

VAPAA

In Finnish, Vapaa means "free", in the sense of "free-range chickens."

Features

- Depends only on Fortran compiler and MPI C library.
  - If MPI C library is compiled without Fortran support, f2c/c2f done by Vapaa.
  - Uses CFI_cdesc_t if possible; can use other array descriptors if necessary.
- Compiles sequentially in less than 2 seconds. Trivial to include in your project.
- Implements all of the mpi_f08 features, even the ridiculous ones.
- Can use MPI C standard ABI to allow MPI Fortran applications to be MPI implementation-agnostic.

Defects

- Implementation incomplete due to developer incompetence and lack of code generation.
- Does not take advantage of MPI implementation properties to optimize f2c/c2f.
- No formal support model. (Feature?)

NVIDIA

Non-cartesian subarrays

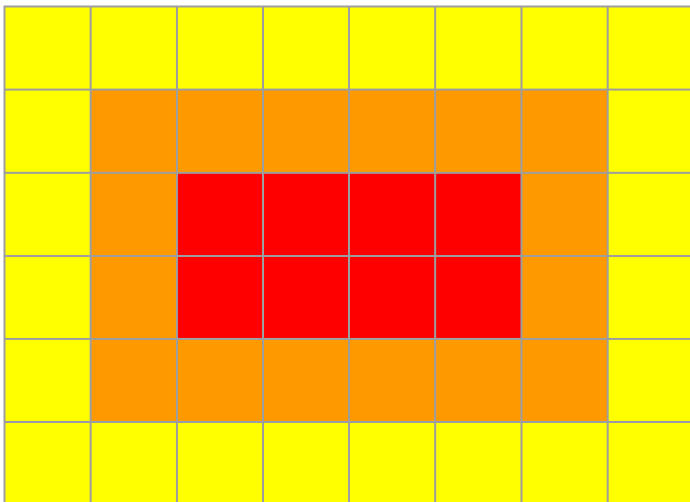
```
integer, dimension(30,20) :: A
integer, dimension(10,10) :: B
...
call MPI_Isend(A(1:30:3,1:20:2), size(A(1:30:3,1:20:2))-1, ..)
call MPI_Irecv(B, size(B), ..)
```

MPI implementations that support subarrays assume cartesian subarrays. The above code is not supported by MPICH, for example.

This use case is fine with blocking communication if the compiler makes a contiguous copy of the input array…

More bad Ideas with MPI_F08

A foolish consistency is the hobgoblin of little minds…



Allocated Fortran array

Fortran subarray buffer passed to MPI

MPI subarray datatype passed to MPI

A(6,8)

B(4,6) = A(2:5,2:7)

C(2,4) = B(2:3,2:5)

It is unlikely that anyone wanted this use case to work, but it was not specifically excluded. It is a natural consequence of two requirements (1) CFI_cdesc_t subarray support and (2) MPI datatypes as they are used in legacy Fortran and C.

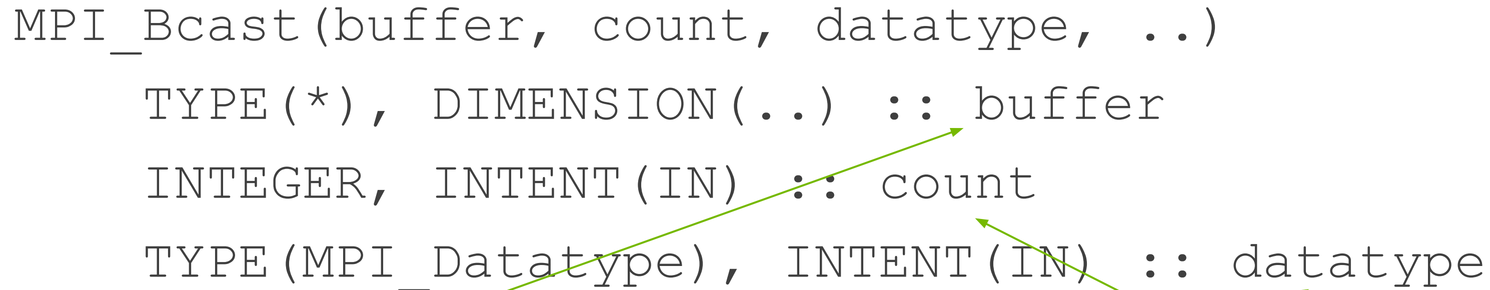
VAPAA

- Specialized support for 1D, 2D, 3D subarrays; generic support for 4D to 15D.
- Support for non-cartesian subarrays.
- Support for subarrays with noncontiguous MPI datatypes.
- Detect inconsistencies between subarray descriptor and MPI count argument.
- Uses MPICH extensions to optimize user-defined datatype decoding.
- Supports CFI_cdesc_t when available; can support other array descriptors; falls back to contiguous-only support.

NVIDIA.

HAVAITA

MPI Fortran API



Describes the base address, element type, element count, and array layout of the user buffer.

Describes the ~~base address,~~ element type, element count, and array layout of the user buffer.

NVIDIA

MPI Fortran API

```
MPI_Bcast(buffer, count, datatype, ..)
    TYPE(*), DIMENSION(..) :: buffer
                           :: count
                INTENT(IN) :: datatype
```

Describes the ~~base address,~~ element type, element count, and array layout of the user buffer.

Smarter MPI Fortran

```
! infer count and datatype from array properties
call MPIX_Bcast(buffer=a1,root=0,comm=world)

! world communicator is implicit
call MPIX_Bcast(buffer=a1,root=0)

! root=0 is implicit
call MPIX_Bcast(buffer=a1,comm=world)

! root=0 and world communicator are implicit
call MPIX_Bcast(buffer=a1)
```

Havaita

Havaita means “detect” or “perceive” in Finnish

- Breaks 1:1 correspondence between C and Fortran API design.
- No need for redundant count and datatype arguments when proper Fortran arrays are used.
- Common cases become implicit.
- MPI_IN_PLACE automatic.
- All arguments are named and placement no longer matters (no need to remember the order, just the names).

https://github.com/jeffhammond/havaita

MPI-Next

MPI Application Binary Interface Standardization

Jeff R. Hammond
NVIDIA Helsinki Oy
Helsinki, Finland

NVHPC SDK, Fortran

Lisandro Dalcin
Extreme Computing Research Center
KAUST
Thuwal, Saudi Arabia
dalc l.com

Python

Erik Schnetter
Perimeter Institute for Theoretical Physics
Waterloo, Ontario, Canada
esc e.ca

Julia, MPItrampoline

Marc Pérache
CEA, DAM, DIF
Arpajon, France

wi4mpi, containers, MPC

Jean-Baptiste Besnard
ParaTools
Bruyères-le-Châtel, France
jbbes s.fr

TAU, E4S

Jed Brown
University of Colorado Boulder
Boulder, Colorado, USA

PETSc, Rust

Gonzalo Brito Gadeschi
NVIDIA GmbH
Munich, Germany

Rust, containers

Joseph Schuchart
University of Tennessee, Knoxville
Knoxville, Tennessee, USA
sc du

Open MPI

Simon Byrne
California Institute of Technology
Pasadena, California, USA
simonby tech.edu

Julia

Hui Zhou
Argonne National Laboratory
Lemont, Illinois, USA
zh ov

MPICH

draft

https://dl.acm.org/doi/10.1145/3615318.3615319

MPI ABI Standardization

- Largely done with the C API ABI.
- Fighting about Fortran-related topics. Fortran does not allow for an ABI, but there are things that make VAPAA and similar easier or harder (f2c/c2f). VAPAA serves as an ABI abstraction layer for **mpi_f08** codes (and maybe MPI legacy Fortran APIs some day).
- CEA's wi4mpi, MPI Trampoline (Erik Schnetter), Mukautuva are ABI translation layers. Mukautuva is a prototype of the standard ABI.
- MPICH has a working prototype of the ABI already, which is actively tested by mpi4py.

Summary

- If you use MPI in Fortran codes, you should use **mpi_f08**.
- VAPAA intends to solve **mpi_f08** availability and feature-completeness issues.
- HAVAITA demonstrates ways to make MPI usage better in Fortran.
- The upcoming MPI standard ABI will make libraries like VAPAA implementation-agnostic. No more MPICH vs Open MPI problems.

https://github.com/jeffhammond/vapaa
https://github.com/jeffhammond/havaita
https://github.com/jeffhammond/mukautuva

NVIDIA