D R A F T
Document for a Standard Message-Passing Interface

Message Passing Interface Forum

April 18, 2013
This work was supported in part by NSF and ARPA under NSF contract CDA-9115428 and Esprit under project HPC Standards (21111).

This is the result of a LaTeX run of a draft of a single chapter of the MPIF Final Report document.

Chapter 17

Language Bindings

17.1 Fortran Support

17.1.1 Overview

...

17.1.2 Fortran Support Through the `mpi_f08` Module

...

17.1.3 Fortran Support Through the `mpi` Module

...

17.1.4 Fortran Support Through the `mpif.h` Include File

...

17.1.5 Interface Specifications, Linker Names and the Profiling Interface

The Fortran interface specifications of each MPI routine specifies the routine name that must be called by the application program, and the names and types of the dummy arguments together with additional attributes. The Fortran standard enables that a given Fortran interface can be implemented with several methods, e.g., within or outside of a module, with or without BIND(C), or the buffers with or without TS29113. Such implementation decisions imply different binary interfaces and different linker names. Several implementation schemes together with the rules for the linker names and its implications for the profiling interface are specified within this section.

Advice to users. The profiling PMPI interface provides the possibility to intercept MPI routines (e.g., MPI_ISEND) by providing an additional MPI_ISEND (the profiling wrapper) that is called by the application and internally calls PMPI_SEND. There are two typical use cases: A profiling layer that is developed independently from the application and the MPI library, and profiling routines that are part of the application and have access to the application data. With MPI-3.0, new Fortran interfaces and implementation schemes were introduced that have several options on how Fortran

MPI routines are internally implemented and optimized. For profiling layers, these schemes imply that several internal interfaces may need to be intercepted. Especially with the implementation scheme B (see below), the interception is done at the MPI C interface, which makes it hard for interception as part of a Fortran application. Therefore, for wrapper routines that are part of a Fortran application it may be more convenient to make the name shift within the application, i.e., to substitute the call to the MPI routine (e.g., MPI_ISEND) by a call to a the user written profiling wrapper with a new name (e.g., X_MPI_ISEND) and to call the Fortran MPI_ISEND from this wrapper. Only for the `mpi` module and the `mpi.h` include file it is still guaranteed to have interceptable Fortran interfaces, see below implementation scheme A. (*End of advice to users.*)

Rationale. This section was introduced in MPI-3.0 on Sep. 21, 2012. The major goals for implementing the three Fortran support methods have been:

- Portable implementation of the wrappers from the MPI Fortran interfaces to the MPI routines in C.
- Binary backward compatible implementation path when switching MPI_SUBARRAYS_SUPPORTED from `.FALSE.` to `.TRUE.`.
- The Fortran PMPI interface need not to be backward compatible, but a method must be included that a tools layer can examine the MPI library about the used linker names and interfaces.
- No performance drawbacks.
- Consistent for all routine groups, as defined below.
- Consistent between all three Fortran support methods.
- Consistent with Fortran 2008 + TS 29113.

The design expected that all dummy arguments in the MPI Fortran interfaces are interoperable with C according to Fortran 2008 + TS 29113. This expectation was not fulfilled. The `LOGICAL` arguments are not interoperable with C, mainly because the values for `.FALSE.` and `.TRUE.` are compiler dependent. The provided interface was mainly based on `BIND(C)` interfaces and therefore inconsistent with Fortran. To be consistent with Fortran, the `BIND(C)` had to be removed from the callback procedure interfaces and the predefined callbacks, see the routine group MPI_COMM_DUP_FN defined below. Non-`BIND(C)` procedures are also not interoperable with C, and therefore the `BIND(C)` had to be removed from all routines with `PROCEDURE` arguments, see the routine groups MPI_OP_CREATE, MPI_REGISTER_DATAREP, and MPI_COMM_CREATE_KEYVAL below.

Therefore, this section was rewritten in an erratum. It defines three implementation schemes. Scheme A removes all `BIND(C)` from the interface definitions. Scheme B provides the rules for implementing the Fortran interface with wrappers that call the MPI routines defined in C. Both schemes can be combined. This combination must be reported to the implementers of profiling layers through special macro definitions in `mpi.h`. For a maximum of backward compatibility, scheme A is required for the `mpi` module and `mpif.h`. Scheme A together with B allows a portable implementation of the Fortran wrappers for all three Fortran support methods and fulfills most of the original goals. A and B are therefore needed to solve the inconsistency problems with the `LOGICAL` arguments.

With the `mpi_f08` module, it is also possible to use scheme B without A for performance reasons, i.e., the wrappers may be contained in the module and may be therefore inlined into the calling application by the compilers.

Scheme C is an additional scheme only for the `mpi_f08` module. It uses `BIND(C)` for the routine groups as long as the dummy arguments are interoperable with C, i.e., with limited consistency. Scheme C is similar to the interface introduced in MPI-3.0 on Sep. 21, 2012.

All schemes include a backward compatibility path for MPI_SUBARRAYS_SUPPORTED through the sub-schemes 1 and 2. For the development of portable profiling layers, all schemes report in `mpi.h` their availability within the corresponding MPI library. (*End of rationale.*)

The linker name of a Fortran routine is defined as the name that a C routine would have if both routines would have the same name visible for the linker. A typical linker name of the Fortran routine FOOfoo is `foofoo__`. In the case of `BIND(C,NAME='...')`, the linker name is directly defined through the external name given by the string.

Similar MPI routines are grouped together for linker symbol scheme classification. If the peer routine of a group is available within an MPI library with one of its possible linker names then all of the routines in this group must be provided according to the same linker name scheme. If the peer routine is not available through a linker name scheme then all other routines in the group must not be available through this scheme. Peer routines and their routine groups are listed Table 17.1 on page 3.

MPI_TEST	All MPI routines that have a LOGICAL dummy argument and that are not callbacks and do not have callback dummy arguments.
MPI_ALLOC_MEM	MPI_ALLOC_MEM, MPI_WIN_ALLOCATE, MPI_WIN_ALLOCATE_SHARED, and MPI_WIN_SHARED_QUERY.
MPI_FREE_MEM	Only this routine is in this group.
MPI_GET_ADDRESS	Only this routine is in this group.
MPI_F_SYNC_REG	Only this routine is in this group.
MPI_SEND	All other routines with choice buffer arguments that are not declared as `ASYNCHRONOUS` within the `mpi_f08` module.
MPI_ISEND	All other routines with choice buffer arguments that are declared as `ASYNCHRONOUS` within the `mpi_f08` module.
MPI_OP_CREATE	Only this routine is in this group.
MPI_REGISTER_DATAREP	Only this routine is in this group.
MPI_COMM_CREATE_KEYVAL	All other routines with callback function arguments.
MPI_COMM_DUP_FN	All predefined callback routines.
MPI_COMM_RANK	All other MPI routines.

Table 17.1: Fortran routine groups.

Advice to implementors. Removed interfaces (see Chapter 16) are in the same routine group as their corresponding replacement functions. (*End of advice to implementors.*)

Different implementation and linker name schemes can be chosen independently for each routine group within each Fortran support method. The Fortran linker names are always based on the routine name (respectively the linker name base) combined with a suffix:

- If the implementation scheme does not use BIND(C) then the linker name mapping of the Fortran compiler is applied. For example, `MPI_Send` together with the suffix `_f08` may be mapped to `mpi_send_f08__`. Prototype example:

```
INTERFACE MPI_Send
  SUBROUTINE MPI_Send_f08(...)
```

- If the implementation scheme uses BIND(C), then the linker name is a combination of the C name and a suffix, e.g., `MPI_Send_f08`. Prototype example:

```
INTERFACE
  SUBROUTINE MPI_Send(...)  BIND(C,NAME='MPI_Send_f08')
```

There are three implementation schemes available:

A. The Fortran MPI routines are implemented outside of a module and without BIND(C). The linker name mapping of the Fortran compiler is applied to a name that is the routine name plus a suffix.

   Special suffixes apply for the three routine groups MPI_SEND, MPI_ISEND, MPI_GET_ADDRESS, and MPI_F_SYNC_REG if they are implemented with `TYPE(*), DIMENSION(..)`, i.e, with TS 29113. The suffixes are listed in columns A1 and A2 in Table 17.2 on page 6.

B. No Fortran linker name is specified, but the Fortran MPI routines must be implemented as (thin) wrappers that call the corresponding C interfaces, see column B1 in Table 17.2 on page 6.

   If `TYPE(*), DIMENSION(..)`, i.e., TS 29113 is used in the routine groups MPI_SEND, MPI_ISEND, MPI_GET_ADDRESS, and MPI_F_SYNC_REG, then the wrappers call a C MPI routine with its name appended with the suffix "`_cdesc`", e.g., `MPI_Send_cdesc`, see column B2. In the interface of these routines, the `void* buffer` arguments are substituted by `const CFI_cdesc_t* buffer`, see TS 29113 [1].

   The wrappers in the routine groups MPI_OP_CREATE, MPI_REGISTER_DATAREP, and MPI_COMM_CREATE_KEYVAL also call C MPI routines with special suffixes, see column B3. Here, the `MPI_xxx_function* xxx_fn` argument is substituted by `void* xxx_fn` to hold the Fortran function pointer or function descriptor pointer, which represent the callback routines with their different interfaces in the `mpi_f08` module, respectively `mpi` and `mpif.h`.

   For profiling, the C MPI interface must be intercepted and the corresponding C PMPI_..._cdesc routines must be called. C PMPI routines must be provided also for the C routines with these special suffixes.

C. The Fortran MPI routines are implemented with BIND(C). The linker name is defined by the corresponding C routine name plus a suffix listed in column C1 in Table 17.2 on page 6.

   Special suffixes apply for the three routine groups MPI_SEND, MPI_ISEND, MPI_GET_ADDRESS, and MPI_F_SYNC_REG if they are implemented with `TYPE(*), DIMENSION(..)`, i.e, with TS 29113, see column C2.

   If the Fortran BIND(C) interface defines a string argument with a fixed size, e.g., `CHARACTER(LEN=xxxx),... :: arg`, then this definition must be substituted by `CHARACTER(LEN=1),DIMENSION(xxxx),... :: arg`. Both definitions have the same meaning for the calling MPI application, but only the second one is interoperable with C (since Fortran 2003).

If a Fortran support method is provided, then at least one of the three schemes must be provided for each routine group within that support method:

- For the Fortran support through the `mpi` module and `mpif.h` it is additionally required that each routine group in `mpif.h` is provided at least with the implementation scheme A. Additionally, the implementation scheme B can be combined with A. Scheme C cannot be applied. Therefore, the entries for scheme C1 and C2 in the last two rows of Table 17.2 are currently not used.
- For the Fortran support through the `mpi_f08` module, an implementor can freely choose between the schemes A, B, and C. Additionally, the implementation scheme B can be combined with A or C.

The implementation scheme A and B are available for all routine groups and all Fortran support methods.

Rationale. Scheme C is not provided for the `mpi` module and `mpif.h` for backward compatibility reasons with MPI-1 and MPI-2. The implementation scheme C can be used only partially in `mpif.h` due to the restriction of 72 characters per line in Fortran fixed source form, and not for the routine group MPI_COMM_DUP_FN nor for those groups with arguments that are not interoperable with C, which are MPI_TEST, MPI_OP_CREATE, MPI_REGISTER_DATAREP, and MPI_COMM_CREATE_KEYVAL in Fortran 2008 + TS 29113 due to `LOGICAL` and `PROCEDURE` arguments in some routine interfaces. (*End of rationale.*)

Rationale. Within each of the columns A1, A2, C1, and C2 of Table 17.2, different suffixes are needed due to the different Fortran interfaces. In B3, different suffixes are needed due to different callback function interfaces. Within each row, different names are needed between all columns; This is guaranteed through the different suffixes and additionally the linker name mapping by the Fortran compiler in columns A1 and A2. (*End of rationale.*)

To set MPI_SUBARRAYS_SUPPORTED to `.TRUE.` within a Fortran support method, it is required that the routine group MPI_ISEND is implemented with A2, B2, or C2.

Several implementation schemes can be included in the MPI library. An MPI library can also include additional schemes that are currently not required by the `mpi_f08` or `mpi` modules or the `mpif.h` include file.

Fortran support methods	Implementation schemes						
	A		B			C	
	A1 normal	A2 TS 29113	B1 normal	B2 TS 29113	B3 special	C1 normal	C2 TS 29113
`mpi_f08`	`_f08`	`_f08ts`	no suffix	`_cdesc`	`_f08cb`	`_f08`	`_f08ts`
`mpi`	no suffix	`_fts`	no suffix	`_cdesc`	`_fcb`	`_f`	`_fts`
`mpif.h`	no suffix	`_fts`	no suffix	`_cdesc`	`_fcb`	`_f`	`_fts`

Table 17.2: Fortran linker name suffixes. The columns "TS 29113" refer only to the routine groups MPI_SEND, MPI_ISEND, MPI_GET_ADDRESS, and MPI_F_SYNC_REG and apply only if TS 29113 is applied for the buffers. The column "special" applies only to the routine groups MPI_OP_CREATE, MPI_REGISTER_DATAREP, and MPI_COMM_CREATE_KEYVAL within the implementation scheme B. The suffixes in all other cases are shown in the columns "normal".

Rationale. After a compiler provides the facilities from TS 29113, i.e., `TYPE(*)`, `DIMENSION(..)`, it is possible to change the bindings within a Fortran support method to support subarrays without recompiling the complete application provided that the previous interfaces are still included in the library. Of course, only recompiled routines can benefit from the added facilities. There is no binary compatibility conflict because each interface uses its own linker names and all interfaces use the same constants and type definitions. (*End of rationale.*)

A user-written or middleware profiling routine that is written according to the same implementation scheme will have the same linker name, and therefore, can interpose itself as the MPI library routine. The profiling routine can internally call the matching PMPI routine with any of its existing bindings, except for routines that have callback routine dummy arguments. In this case, the profiling software must use the same Fortran support method as used in the calling application program, because the C, `mpi_f08` and `mpi` callback prototypes are different.

In the case that a Fortran binding consists of multiple routines through function overloading, the base names of overloaded routines are appended by a suffix indicating the difference in the argument list. For example, MPI_ALLOC_MEM (in the `mpi` module and `mpif.h`) has an `INTEGER(KIND=...) baseptr` argument without a suffix. This routine is overloaded by a routine with `TYPE(C_PTR) baseptr` and the suffix `_CPTR`. The implied linker name base is MPI_ALLOC_MEM_CPTR. It is mapped to the linker names `MPI_Alloc_mem_cptr_f` (in C1), and, e.g., `mpi_alloc_mem_cptr__` (in A1). Note that these routines are always called via the interface name MPI_ALLOC_MEM by the application within all Fortran support methods.

Additionally, several C preprocessor macros are available in `mpi.h` for each routine group. The name of the macros are the peer routine name written as in Table 17.1 on page 3 and appended with two suffixes.

First suffix:

_mpi_f08	To report the implementation scheme(s) used in the `mpi_f08` module.
_mpi	To report the implementation scheme(s) used in the `mpi` module.
_mpifh	To report the implementation scheme(s) used in the `mpif.h` include file.

Second suffix:

_A1	Available for all routine groups.
_A2	Available only for the routine groups MPI_SEND, MPI_ISEND, MPI_GET_ADDRESS, and MPI_F_SYNC_REG.
_B1	Available for all routine groups, except for the routine groups MPI_OP_CREATE, MPI_REGISTER_DATAREP, and MPI_COMM_CREATE_KEYVAL.
_B2	Available only for the routine groups MPI_SEND, MPI_ISEND, MPI_GET_ADDRESS, and MPI_F_SYNC_REG.
_B3	Available only for the routine groups MPI_OP_CREATE, MPI_REGISTER_DATAREP, and MPI_COMM_CREATE_KEYVAL.
_C1	Available for all routine groups, except the routine group MPI_COMM_DUP_FN.
_C2	Available only for the routine groups MPI_SEND, MPI_ISEND, MPI_GET_ADDRESS, and MPI_F_SYNC_REG.

If a combination of a Fortran support method and an implementation scheme can be used for a routine group then the appropriate macro must be available, e.g., `MPI_TEST_mpi_f08_A1`. If an implementation scheme is not available for a routine group the macro must not be available, e.g., `MPI_TEST_mpi_f08_A2`.

The value of the macros indicate:

0	The Routine group is not available in this Fortran support method with this implementation scheme.
1	Available in the MPI library but not used in the module or include file.
2	Available in the MPI library and used in the module or include file.

In the examples in Table 17.3 on page 8, the values in column “A1+A2” show that

- the routines in the MPI_SEND group are only available through their Fortran linker names (e.g., `mpi_send_f08__`, `mpi_send__`, `mpi_recv_f08__`, `mpi_recv__`, ...),
- the routines in the MPI_ISEND group are available through several interfaces: a call to MPI_ISEND
  - with the `mpi_f08` module is mapped to, e.g., `mpi_isend_f08ts__`,
  - with the `mpi` module is mapped to, e.g., `mpi_isend_fts__`,
  - with the `mpif.h` include file is mapped to, e.g., `mpi_isend_fts__`.

  All three Fortran support methods provide TS 29113 quality, i.e., MPI_SUBARRAYS_SUPPORTED equals `.TRUE.`. The MPI library additional contains, e.g., `mpi_isend_f08__`, `mpi_isend__` to support binary applications that were compiled with an older MPI library version with MPI_ISEND_mpi_f08_A1, MPI_ISEND_mpi_A1, and MPI_ISEND_mpifh_A1 set to 2, and MPI_ISEND_mpi_f08_A2, MPI_ISEND_mpi_A2, and MPI_ISEND_mpifh_A2 set to 0.

Note that for the routines with callback procedure arguments, e.g., in the routine group MPI_OP_CREATE, the macros `..._B1` are substituted by `..._B3`. For the predefined callbacks, the implementation scheme C1 does not exist because the interfaces must fit to the

Used implementation schemes:	A1+A2	A1 +B1	A1/2 +B1/2	A1+A2 +B1+B2
Implied values for MPI_SUBARRAYS_SUPPORTED:	.TRUE.	.FALSE.	.TRUE.	.TRUE.
/* Values for the Fortran support method with the mpi_f08 module */				
#define MPI_TEST_mpi_f08_A1	2	2	2	2
#define MPI_TEST_mpi_f08_B1	0	2	2	2
#define MPI_TEST_mpi_f08_C1	0	0	0	0
... (same for routine groups MPI_ALLOC_MEM and MPI_FREE_MEM)				
#define MPI_SEND_mpi_f08_A1	2	2	0	1
#define MPI_SEND_mpi_f08_A2	0	0	2	2
#define MPI_SEND_mpi_f08_B1	0	2	0	1
#define MPI_SEND_mpi_f08_B2	0	0	2	2
#define MPI_SEND_mpi_f08_C1	0	0	0	0
#define MPI_SEND_mpi_f08_C2	0	0	0	0
... (same for routine groups MPI_GET_ADDRESS and MPI_F_SYNC_REG)				
#define MPI_ISEND_mpi_f08_A1	1	2	0	1
#define MPI_ISEND_mpi_f08_A2	2	0	2	2
#define MPI_ISEND_mpi_f08_B1	0	2	0	1
#define MPI_ISEND_mpi_f08_B2	0	0	2	2
#define MPI_ISEND_mpi_f08_C1	0	0	0	0
#define MPI_ISEND_mpi_f08_C2	0	0	0	0
#define MPI_OP_CREATE_mpi_f08_A1	2	2	2	2
#define MPI_OP_CREATE_mpi_f08_B3	0	2	2	2
#define MPI_OP_CREATE_mpi_f08_C1	0	0	0	0
... (same for MPI_REGISTER_DATAREP and MPI_COMM_CREATE_KEYVAL)				
#define MPI_COMM_DUP_FN_mpi_f08_A1	2	2	2	2
#define MPI_COMM_DUP_FN_mpi_f08_B1	0	2	2	2
#define MPI_COMM_RANK_mpi_f08_A1	2	2	2	2
#define MPI_COMM_RANK_mpi_f08_B1	0	2	2	2
#define MPI_COMM_RANK_mpi_f08_C1	0	0	0	0
/* Values for the Fortran support method with the mpi module */ ... (and the same values for macros MPI_XXX_mpi_A/B/C.)				
/* Values for the Fortran support method with mpif.h */ ... (and the same values for macros MPI_XXX_mpifh_A/B/C.)				

Table 17.3: C preprocessor macros and possible values.

callback function prototypes, which are defined without `BIND(C)` in all Fortran support methods.

Whithin each macro block that contains the macros for one routine group with one Fortran support method, one macro must have the value 2. A second macro can be set to 2 only if A or C is implemented with (thin) wrappers according to B. All other macros must be set to 1 or 0.

The column "A1+B1" reflects an implementation without TS 29113 that uses the schemes A and B together. The column "A1/2+B1/2" uses A and B and provides full TS 29113 quality, but the old A1 and B1 binaries are removed from the library.

Advice to implementors. If all following conditions are fulfilled (which is the case for most compilers):

- the handles in the `mpi_f08` module occupy one Fortran numerical storage unit (same as an `INTEGER` handle),
- the internal argument passing mechanism used to pass an actual ierror argument to a non-optional ierror dummy argument is binary compatible to passing an actual ierror argument to an ierror dummy argument that is declared as `OPTIONAL`,
- the internal argument passing mechanism for `ASYNCHRONOUS` and non-`ASYNCHRONOUS` arguments is the same,
- the internal routine call mechanism is the same for the Fortran and the C compilers for which the MPI library is compiled,
- the compiler does not provide TS 29113,

then for the routine groups, the implementor may use the same internal routine implementations for all Fortran support methods but with several different linker names. For TS 29113 quality, new routines are needed only for the routine group of MPI_ISEND. (*End of advice to implementors.*)

Advice to implementors. The implementation scheme A1 and A2 can be also implemented for all routines in the Fortran support method `mpif.h` with compile-time argument checking. For `mpif.h`, the argument names are not specified through the MPI standard, i.e., only positional argument lists are defined, and not key-word based lists. Due to the rule that `mpif.h` must be valid for fixed and free source form, the subroutine declaration is restricted to one line with 72 characters. To keep the argument lists short, each argument name can be shortened to a minimum of one character. With this, the two longest subroutine declaration statements in A1 are

```
SUBROUTINE PMPI_Dist_graph_create_adjacent(a,b,c,d,e,f,g,h,i,j,k)
SUBROUTINE PMPI_Rget_accumulate(a,b,c,d,e,f,g,h,i,j,k,l,m,n)
```

with 71 and 66 characters, and the longest interface definition in A2 is

```
INTERFACE  PMPI_Rget_accumulate
SUBROUTINE PMPI_Rget_accumulate_fts(a,b,c,d,e,f,g,h,i,j,k,l,m,n)
```

with 70 characters. In principle, continuation lines would be possible in mpif.h (spaces on columns 73-131, & on column 132, and on column 5 of the continuation line) but this would not be valid if the source line length is extended with a compiler flag to 132 characters. Column 133 is also not available for the continuation character because lines longer than 132 characters are invalid with some compilers by default.

The longest linker name in A1 is derived from PMPI_Dist_graph_create_adjacent_f08 and in A2 from PMPI_File_write_ordered_begin_f08ts both with 35 characters in the `mpi_f08` module.

The implementation scheme B the longest linker names is PMPI_File_write_ordered_begin_cdesc in B2 with 35 characters, and PMPI_Comm_create_errhandler_f08cb in B3 with 33 characters.

The implementation scheme A together with B can be implemented with portable wrappers. Routine interface declarations are:

```
MODULE mpi_f08
  INTERFACE MPI_Comm_rank  ! (as defined in Chapter 6)
    SUBROUTINE MPI_Comm_rank_f08(comm, rank, ierror)
      TYPE(MPI_Comm),      INTENT(IN)  :: comm
      INTEGER,             INTENT(IN)  :: rank
      INTEGER, OPTIONAL,   INTENT(OUT) :: ierror
    END SUBROUTINE
  END INTERFACE
END MODULE mpi_f08

MODULE mpi
  INTERFACE MPI_Comm_rank  ! (as defined in Chapter 6)
    SUBROUTINE MPI_Comm_rank(comm, rank, ierror)
      INTEGER, INTENT(IN)  :: comm   ! The INTENT may be added although
      INTEGER, INTENT(IN)  :: rank   ! it is not defined in the
      INTEGER, INTENT(OUT) :: ierror ! official routine definition.
    END SUBROUTINE
  END INTERFACE
END MODULE mpi
```

And probably in `mpif.h` (outside of any module) and in fixed source format:

```
!23456789012345678901234567890123456789012345678901234567890123456789012
      INTERFACE MPI_Comm_rank  ! (as defined in Chapter 6)
       SUBROUTINE MPI_Comm_rank(comm, rank, ierror)
        INTEGER, INTENT(IN)  :: comm   ! The argument names may be
        INTEGER, INTENT(IN)  :: rank   ! shortened that the subroutine
        INTEGER, INTENT(OUT) :: ierror ! line fits to the maximum
       END SUBROUTINE                  ! of 72 characters.
      END INTERFACE
```

The Fortran declaration for the existing MPI C library routine is:

```
MODULE mpi_X_C_interfaces
  INTERFACE
    FUNCTION MPI_X_Comm_rank_C(comm,rank) &
               BIND(C, name='MPI_Comm_rank') RESULT(ierror_c)
      USE,INTRINSIC :: iso_c_binding, only: c_int
      USE :: mpi_X_FC_wrappers, only:  MPI_C_COMM_KIND
      INTEGER(KIND=MPI_C_COMM_KIND),VALUE,INTENT(IN) :: comm
      INTEGER(KIND=c_int),INTENT(OUT)                :: rank
      INTEGER(KIND=c_int)                            :: ierror_c
    END FUNCTION
  END INTERFACE
END MODULE mpi_X_C_interfaces
```

The implementation of the Fortran MPI_Comm_rank subroutine for the mpi_f08 module as a wrapper to the C routine is:

```
SUBROUTINE MPI_Comm_rank_f08(comm, rank, ierror)
  USE,INTRINSIC :: iso_c_binding, only: c_int
  USE :: mpi_X_f08_types, only:    MPI_Comm
  USE :: mpi_X_FC_wrappers, only:  MPI_C_COMM_KIND, MPI_Comm_f2c
  USE :: mpi_X_C_interfaces, only: MPI_X_Comm_rank_C
  TYPE(MPI_Comm),      INTENT(IN)  :: comm
  INTEGER,             INTENT(IN)  :: rank
  INTEGER, OPTIONAL,   INTENT(OUT) :: ierror
  INTEGER(KIND=MPI_C_COMM_KIND)    :: comm_c
  INTEGER(KIND=c_int)              :: rank_c
  INTEGER(KIND=c_int)              :: ierror_c
  comm_c = MPI_Comm_f2c(comm%MPI_VAL)
  ierror_c = MPI_X_Comm_rank_C(comm_c, rank_c)
  rank = rank_c
  IF (PRESENT(ierror)) ierror = ierror_c
END SUBROUTINE
```

The implementation of the Fortran MPI_COMM_RANK subroutine for the mpi module and mpif.h as a wrapper to the C routine is:

```
SUBROUTINE MPI_COMM_RANK(comm, rank, ierror)
  ...
  INTEGER, INTENT(IN)  :: comm
  ...
  comm_c = MPI_Comm_f2c(comm)
  ierror_c = MPI_X_Comm_rank_C(comm_c, rank_c)
  rank = rank_c
  ierror = ierror_c
END SUBROUTINE
```

If these wrapper subroutines are implemented outside of the mpi_f08 and mpi modules, i.e., without CONTAINS, then this implementation is a valid implementation according to scheme A. It is also a valid scheme B implementation because the call to the interface MPI_X_Comm_rank_C is mapped to a call to the C routine MPI_Comm_rank as specified in the mpi_X_C_interfaces module.

Scheme B can also be used together with scheme C (instead of A). In this case, the interface and the wrapper routine of MPI_Comm_rank are defined as BIND(C), and in the mpi module and mpif.h, the routine name is appended with the suffix _f, and in the mpi_f08 module with the suffix _f08. The interface names are always MPI_Comm_rank, and due to BIND(C), the compiler's name mangling does not apply.

For the mpi module and mpif.h, the implementation must be outside of a module. If the mpi_f08 module CONTAINS the wrapper subroutine then the compiler may inline the wrapper code into the application and the application binary directly calls the MPI C routine without any additional subroutine call overhead for this wrapper. In this case, implementation scheme A does not apply because the compiler uses an internal name mangling for the linker names.

The module names `mpi_X_f08_types`, `mpi_X_C_wrappers`, and `mpi_X_C_interfaces`, and the function name MPI_X_Comm_rank_C are implementation dependent.

For subroutines with buffers, the scheme B2 applies if TS 29113 is available. Without TS 29113, B1 applies.

Here an example with A2 together with B2. Routine interface declarations are:

```
MODULE mpi_f08
  INTERFACE MPI_Irecv ! (arguments as defined in Chapter 3)
    SUBROUTINE MPI_Irecv_f08ts(buf,count,datatype,source,tag,comm,request,ierror)
      TYPE(*), DIMENSION(..), ASYNCHRONOUS :: buf
      ...
    END SUBROUTINE
  END INTERFACE
END MODULE mpi_f08
```

The corresponding C interface definition of the MPI C library function is:

```
#include "ISO_Fortran_binding.h"
int MPI_Irecv_cdesc(const CFI_cdesc_t *buf, int count, MPI_Datatype datatype,
                     int source, int tag, MPI_Comm comm, MPI_Request *request)
```

This C library routine exists only if B2 is used. The corresponding Fortran declaration of the C library function is:

```
MODULE mpi_X_C_interfaces
  INTERFACE
    FUNCTION MPI_Irecv_cdesc(buf,count,datatype,source,tag,comm,request) &
                BIND(C, name='MPI_Irecv_cdesc') RESULT(ierror_c)
      TYPE(*),DIMENSION(..),ASYNCHRONOUS     :: buf
      ...
    END FUNCTION
  END INTERFACE
END MODULE mpi_X_C_interfaces
```

The wrapper subroutine is:

```
SUBROUTINE MPI_Irecv_f08ts(buf,count,datatype,source,tag,comm,request,ierror)
  TYPE(*), DIMENSION(..), ASYNCHRONOUS :: buf
  ...
  ierror_c=MPI_Irecv_cdesc(buf,count_c,datatype_c,source_c,tag_c,comm_c,request_c)
  request%MPI_VAL = request_c
  if (present(ierror)) ierror = ierror_c
END SUBROUTINE
```

Note that the MPI_Irecv_f08ts routine is not contained in the module, i.e., the example fulfills the naming rules for A. In the `mpi` module and in `mpif.h`, the routine name MPI_Irecv_f08ts has to be substituted by MPI_Irecv_fts.

If the mpi module or mpif.h does not provide TYPE(*), DIMENSION(..) for choice buffers, or if the mpi_f08 module is preliminarily implemented without TS 29113 then the operation can be implemented according to A1 and B1 with the following nterface specifications and wrapper routine:

```
MODULE mpi_f08
  INTERFACE MPI_Irecv ! (arguments as defined in Chapter 3)
    SUBROUTINE MPI_Irecv_f08(buf,count,datatype,source,tag,comm,request,ierror)
      !non-standardized declaration of buf, e.g.,
      ! TYPE(*), DIMENSION(..), ASYNCHRONOUS :: buf
        !DEC$ ATTRIBUTES NO_ARG_CHECK :: buf
        !$PRAGMA IGNORE_TKR buf
        !DIR$ IGNORE_TKR buf
        !IBM* IGNORE_TKR buf
        INTEGER, DIMENSION(*), ASYNCHRONOUS :: buf ! choice-dummy-argument
      ...
    END SUBROUTINE
  END INTERFACE
END MODULE mpi_f08
```

Note that if such non-standard extensions are not provided by the compiler then exceptions apply, i.e., within the mpi module and mpif.h, implicit interfaces must be used, and it is recommended to provide the mpi_f08 module only if TS 29113 or such extensions exist, for further details see Section 17.1.6 on page 14. Note that TYPE(*), DIMENSION(*) must not be used because it does not support actual arguments that are non-array variables. Overloading with a second interface with a non-array buffer is possible, but would prevent profiling through the scheme A1.

The corresponding Fortran declaration of the C library function MPI_Irecv is:

```
MODULE mpi_X_C_interfaces
  INTERFACE
    FUNCTION MPI_X_Irecv_C(buf,count,datatype,source,tag,comm,request) &
                BIND(C, name='MPI_Irecv') RESULT(ierror_c)
      TYPE(*),DIMENSION(*),ASYNCHRONOUS :: buf
      ! which may be substituted (if TS 29113 is not available) by:
      ! INTEGER,DIMENSION(*),ASYNCHRONOUS :: buf
      ...
    END FUNCTION
  END INTERFACE
END MODULE mpi_X_C_interfaces
```

The wrapper subroutine is:

```
SUBROUTINE MPI_Irecv_f08(buf,count,datatype,source,tag,comm,request,ierror)
 !non-standardized declaration of buf, e.g.,
 ...
 ierror_c=MPI_X_Irecv_C(buf,count_c,datatype_c,source_c,tag_c,comm_c,request_c)
 ...
END SUBROUTINE
```

In the mpi module and in mpif.h, the routine name MPI_Irecv_f08 has to be substituted by MPI_Irecv.

Special care must be taken in the wrapper routines for arguments of type `LOGICAL` and `CHARACTER`. With `LOGICAL` arguments the non-standardized binary representations of `.TRUE.` and `.FALSE.` must be mapped to 1 and 0 in C. Character string arguments must be converted between the space filled strings with explicit length information in Fortran and the `\0`-terminated strings in C.

(*End of advice to implementors.*)

17.1.6 MPI for Different Fortran Standard Versions

...

17.1.7 Requirements on Fortran Compilers

...

...

...

17.2 Language Interoperability

...

17.2.1 Introduction

...

17.2.2 Assumptions

...

17.2.3 Initialization

...

17.2.4 Transfer of Handles

...

Within the `mpi_f08` and `mpi` modules and `mpif.h`, additional Fortran interfaces are defined with BIND(C) to access the C wrappers defined within this section, for example:

```
INTERFACE
  FUNCTION MPI_Comm_f2c(comm) &
              BIND(C, name='MPI_Comm_f2c') RESULT(comm_c)
    INTEGER,VALUE,INTENT(IN)      :: comm
    INTEGER(KIND=MPI_C_COMM_KIND) :: comm_c
  END FUNCTION
END INTERFACE
```

The Fortran kind-specification MPI_C_COMM_KIND reflects the integer size needed to store a C MPI_Comm handle. Other C handle kind parameters are MPI_C_DATATYPE_KIND, MPI_C_ERRHANDLER_KIND, MPI_C_FILE_KIND, MPI_C_GROUP_KIND, MPI_C_INFO_KIND, MPI_C_MESSAGE_KIND, MPI_C_OP_KIND, MPI_C_REQUEST_KIND, and MPI_C_WIN_KIND. They are available in Fortran only, and there with all Fortran support methods. In the same way, interfaces must be provided for the PMPI_..._f2c and PMPI_..._c2f routines.

Within the `mpi_f08` and `mpi` modules, an MPI implementor can choose to provide these interfaces with the same function name directly, i.e., without access to the C function, and defined with or without BIND(C). Such implementations may be inlined into the application binary. In this case, interception for profiling is not provided.

17.2.5 Status

...

17.2.6 MPI Opaque Objects

...

17.2.7 Attributes

...

17.2.8 Extra-State

...

17.2.9 Constants

...

17.2.10 Interlanguage Communication

...

Annex A

Language Bindings Summary

A.1 Defined Values and Handles

A.1.1 Defined Constants

...

Variable Address Size (Fortran only)
Fortran type: `INTEGER`
MPI_ADDRESS_KIND
MPI_COUNT_KIND
MPI_INTEGER_KIND
MPI_OFFSET_KIND
MPI_C_COMM_KIND
MPI_C_DATATYPE_KIND
MPI_C_ERRHANDLER_KIND
MPI_C_FILE_KIND
MPI_C_GROUP_KIND
MPI_C_INFO_KIND
MPI_C_MESSAGE_KIND
MPI_C_OP_KIND
MPI_C_REQUEST_KIND
MPI_C_WIN_KIND

Bibliography

[1] International Organization for Standardization, ISO/IEC/SC22/WG5 (Fortran), Geneva, TS 29113. *TS on further interoperability with C*, 2012. http://www.nag.co.uk/sc22wg5/, successfully balloted DTS at ftp://ftp.nag.co.uk/sc22wg5/N1901-N1950/N1917.pdf. 17.1.5

Index