

D R A F T

Document for a Standard Message-Passing Interface

Message Passing Interface Forum

April 17, 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

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

12
 13 *Rationale.* This section was introduced in MPI-3.0 on Sep. 21, 2012. The major
 14 goals for implementing the three Fortran support methods have been:

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

25
 26
 27 The design expected that all dummy arguments in the MPI Fortran interfaces are inter-
 28 operable with C according to Fortran 2008 + TS 29113. This expectation was not
 29 fulfilled. The `LOGICAL` arguments are not interoperable with C, mainly because the
 30 values for `.FALSE.` and `.TRUE.` are compiler dependent. The provided interface was
 31 mainly based on `BIND(C)` interfaces and therefore inconsistent with Fortran. To be
 32 consistent with Fortran, the `BIND(C)` had to be removed from the callback procedure
 33 interfaces and the predefined callbacks, see the routine group `MPI_COMM_DUP_FN`
 34 defined below. Non-`BIND(C)` procedures are also not interoperable with C, and
 35 therefore the `BIND(C)` had to be removed from all routines, see the routine groups
 36 `MPI_OP_CREATE`, `MPI_REGISTER_DATAREP`, and `MPI_COMM_CREATE_KEYVAL`
 37 below.

38
 39 Therefore, this section was rewritten in an erratum. It defines three implementation
 40 schemes. Scheme A removes all `BIND(C)` from the interface definitions. Scheme B
 41 provides the rules for implementing the Fortran interface with wrappers that call the
 42 MPI routines defined in C. Both schemes can be combined. This combination must be
 43 reported to the implementers of profiling layers through special macro definitions in
 44 `mpif.h`. For a maximum of backward compatibility, scheme A is required for the `mpi`
 45 module and `mpif.h`. Scheme A together with B allows a portable implementation of
 46 the Fortran wrappers for all three Fortran support methods and fulfills most of the
 47 original goals. A and B are therefore needed to solve the inconsistency problems with
 48 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 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. 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.

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

Table 17.1: Fortran routine groups.

1 *Advice to implementors.* Removed interfaces (see Chapter 16) are in the same routine
 2 group as their corresponding replacement functions. (*End of advice to implementors.*)
 3

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

- 8 • If the implementation scheme does not use BIND(C) then the linker name mapping
 9 of the Fortran compiler is applied. For example, MPI_Send together with the suffix
 10 _f08 may be mapped to mpi_send_f08__. Prototype example:
 11

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

- 14 • If the implementation scheme uses BIND(C), then the linker name is a combination
 15 of the C name and a suffix, e.g., MPI_Send_f08. Prototype example:
 16

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

19 There are three implementation schemes available:
 20

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

24 Special suffixes apply for the three routine groups MPI_SEND, MPI_ISEND,
 25 MPI_GET_ADDRESS, and MPI_F_SYNC_REG if they are implemented with TYPE(*),
 26 DIMENSION(. .), i.e, with TS 29113. The suffixes are listed in columns A1 and A2 in
 27 Table 17.2 on page 6.
 28

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

32 If TYPE(*), DIMENSION(. .), i.e., TS 29113 is used in the routine groups MPI_SEND,
 33 MPI_ISEND, MPI_GET_ADDRESS, and MPI_F_SYNC_REG, then the wrappers call a
 34 C MPI routine with its name appended with the suffix “_cdesc”, e.g., MPI_Send_cdesc,
 35 see column B2. In the interface of these routines, the void* buffer arguments are
 36 substituted by const CFI_cdesc_t* buffer, see TS 29113 [1].
 37

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

44 For profiling, the C MPI interface must be intercepted and the corresponding C
 45 PMPI_..._cdesc routines must be called. C PMPI routines must be provided also for
 46 the C routines with these special suffixes.
 47
 48

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	A2	B1	B2	B3	C1	C2
	normal	TS 29113	normal	TS 29113	special	normal	TS 29113
<code>mpi_f08</code>	<code>_f08</code>	<code>_f08ts</code>	no suffix	<code>_cdesc</code>	<code>_f08cb</code>	<code>_f08</code>	<code>_f08ts</code>
<code>mpi</code>	no suffix	<code>_fts</code>	no suffix	<code>_cdesc</code>	<code>_fcb</code>	<code>_f</code>	<code>_fts</code>
<code>mpif.h</code>	no suffix	<code>_fts</code>	no suffix	<code>_cdesc</code>	<code>_fcb</code>	<code>_f</code>	<code>_fts</code>

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_mpfih_A1` set to 2, and `MPI_ISEND_mpi_f08_A2`, `MPI_ISEND_mpi_A2`, and `MPI_ISEND_mpfih_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_mpfifh_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

1 this would not be valid if the source line length is extended with a compiler flag to 132
 2 characters. Column 133 is also not available for the continuation character because
 3 lines longer than 132 characters are invalid with some compilers by default.

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

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

10 The implementation scheme A together with B can be implemented with portable
 11 wrappers. Routine interface declarations are:
 12

```

13 MODULE mpi_f08
14   INTERFACE MPI_Comm_rank ! (as defined in Chapter 6)
15     SUBROUTINE MPI_Comm_rank_f08(comm, rank, ierror)
16       TYPE(MPI_Comm),      INTENT(IN)  :: comm
17       INTEGER,              INTENT(IN)  :: rank
18       INTEGER, OPTIONAL,    INTENT(OUT) :: ierror
19     END SUBROUTINE
20   END INTERFACE
21 END MODULE mpi_f08

22 MODULE mpi
23   INTERFACE MPI_Comm_rank ! (as defined in Chapter 6)
24     SUBROUTINE MPI_Comm_rank(comm, rank, ierror)
25       INTEGER, INTENT(IN)  :: comm ! The INTENT may be added although
26       INTEGER, INTENT(IN)  :: rank ! it is not defined in the
27       INTEGER, INTENT(OUT) :: ierror ! official routine definition.
28     END SUBROUTINE
29   END INTERFACE
30 END MODULE mpi

```

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

```

32
33 !234567890123456789012345678901234567890123456789012345678901234567890123456789012
34   INTERFACE MPI_Comm_rank ! (as defined in Chapter 6)
35     SUBROUTINE MPI_Comm_rank(comm, rank, ierror)
36       INTEGER, INTENT(IN)  :: comm ! The argument names may be
37       INTEGER, INTENT(IN)  :: rank ! shortened that the subroutine
38       INTEGER, INTENT(OUT) :: ierror ! line fits to the maximum
39     END SUBROUTINE
40   END INTERFACE

```

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

```

42
43 MODULE mpi_X_C_interfaces
44   INTERFACE
45     FUNCTION MPI_X_Comm_rank_C(comm,rank) &
46       BIND(C, name='MPI_Comm_rank') RESULT(ierror_c)
47     USE, INTRINSIC :: iso_c_binding, only: c_int
48     USE :: mpi_X_FC_wrappers, only: MPI_C_COMM_KIND
49     INTEGER(KIND=MPI_C_COMM_KIND), VALUE, INTENT(IN) :: comm

```

```

        INTEGER(KIND=c_int),INTENT(OUT)           :: rank           1
        INTEGER(KIND=c_int)                       :: ierror_c      2
    END FUNCTION                                           3
END INTERFACE                                           4
END MODULE mpi_X_C_interfaces                             5

```

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)           9
    USE,INTRINSIC :: iso_c_binding, only: c_int           10
    USE :: mpi_X_f08_types, only: MPI_Comm                11
    USE :: mpi_X_FC_wrappers, only: MPI_C_COMM_KIND, MPI_Comm_f2c 12
    USE :: mpi_X_C_interfaces, only: MPI_X_Comm_rank_C    13
    TYPE(MPI_Comm),          INTENT(IN)  :: comm           14
    INTEGER,                 INTENT(IN)  :: rank           15
    INTEGER, OPTIONAL,      INTENT(OUT)  :: ierror        16
    INTEGER(KIND=MPI_C_COMM_KIND) :: comm_c              17
    INTEGER(KIND=c_int)      :: rank_c                  18
    INTEGER(KIND=c_int)      :: ierror_c                19
    comm_c = MPI_Comm_f2c(comm%MPI_VAL)                 20
    ierror_c = MPI_X_Comm_rank_C(comm_c, rank_c)         21
    rank = rank_c                                       22
    IF (PRESENT(ierror)) ierror = ierror_c              23
END SUBROUTINE                                           24

```

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)              25
    ...                                                    26
    INTEGER, INTENT(IN)  :: comm                          27
    ...                                                    28
    comm_c = MPI_Comm_f2c(comm)                          29
    ierror_c = MPI_X_Comm_rank_C(comm_c, rank_c)          30
    rank = rank_c                                         31
    ierror = ierror_c                                    32
END SUBROUTINE                                           33

```

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

1 inline the wrapper code into the application and the application binary directly calls
 2 the MPI C routine without any additional subroutine call overhead for this wrapper.
 3 In this case, implementation scheme A does not apply because the compiler uses an
 4 internal name mangling for the linker names.

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

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

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

```
10
11 MODULE mpi_f08
12   INTERFACE MPI_Irecv ! (arguments as defined in Chapter 3)
13     SUBROUTINE MPI_Irecv_f08ts(buf,count,datatype,source,tag,comm,request,ierror)
14       TYPE(*), DIMENSION(..), ASYNCHRONOUS :: buf
15       ...
16     END SUBROUTINE
17   END INTERFACE
18 END MODULE mpi_f08
```

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

```
20
21 #include "ISO_Fortran_binding.h"
22 int MPI_Irecv_cdesc(const CFI_cdesc_t *buf, int count, MPI_Datatype datatype,
23                    int source, int tag, MPI_Comm comm, MPI_Request *request)
24
```

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

```
27
28 MODULE mpi_X_C_interfaces
29   INTERFACE
30     FUNCTION MPI_Irecv_cdesc(buf,count,datatype,source,tag,comm,request) &
31       BIND(C, name='MPI_Irecv_cdesc') RESULT(ierror_c)
32     TYPE(*),DIMENSION(..),ASYNCHRONOUS      :: buf
33     ...
34   END FUNCTION
35 END INTERFACE
36 END MODULE mpi_X_C_interfaces
```

37 The wrapper subroutine is:

```
38
39 SUBROUTINE MPI_Irecv_f08ts(buf,count,datatype,source,tag,comm,request,ierror)
40   TYPE(*), DIMENSION(..), ASYNCHRONOUS :: buf
41   ...
42   ierror_c=MPI_Irecv_cdesc(buf,count_c,datatype_c,source_c,tag_c,comm_c,request_c)
43   request%MPI_VAL = request_c
44   if (present(ierror)) ierror = ierror_c
45 END SUBROUTINE
```

46 Note that the `MPI_Irecv_f08ts` routine is not contained in the module, i.e., the example
 47 fulfills the naming rules for A. In the `mpi` module and in `mpif.h`, the routine name
 48 `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 interface 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`.

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

6 *(End of advice to implementors.)*
 7
 8

9 17.1.6 MPI for Different Fortran Standard Versions

10 ...
 11

12 17.1.7 Requirements on Fortran Compilers

13 ...
 14

15 ...
 16

17 ...
 18

19 17.2 Language Interoperability

20 ...
 21

22 17.2.1 Introduction

23 ...
 24

25 17.2.2 Assumptions

26 ...
 27

28 17.2.3 Initialization

29 ...
 30

31 17.2.4 Transfer of Handles

32 ...
 33

34 Within the `mpi_f08` and `mpi` modules and `mpif.h`, additional Fortran interfaces are defined
 35 with `BIND(C)` to access the C wrappers defined within this section, for example:
 36

```
37
38
39 INTERFACE
40   FUNCTION MPI_Comm_f2c(comm) &
41     BIND(C, name='MPI_Comm_f2c') RESULT(comm_c)
42     INTEGER, VALUE, INTENT(IN)      :: comm
43     INTEGER(KIND=MPI_C_COMM_KIND) :: comm_c
44   END FUNCTION
45 END INTERFACE
46
47
48
```


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

...

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48

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

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48

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

CONST:MPI_ADDRESS_KIND, 16
CONST:MPI_C_COMM_KIND, 15, 16
CONST:MPI_C_DATATYPE_KIND, 15, 16
CONST:MPI_C_ERRHANDLER_KIND, 15, 16
CONST:MPI_C_FILE_KIND, 15, 16
CONST:MPI_C_GROUP_KIND, 15, 16
CONST:MPI_C_INFO_KIND, 15, 16
CONST:MPI_C_MESSAGE_KIND, 15, 16
CONST:MPI_C_OP_KIND, 15, 16
CONST:MPI_C_REQUEST_KIND, 15, 16
CONST:MPI_C_WIN_KIND, 15, 16
CONST:MPI_COMM_DUP_FN, 3
CONST:MPI_COUNT_KIND, 16
CONST:MPI_INTEGER_KIND, 16
CONST:MPI_OFFSET_KIND, 16
CONST:MPI_SUBARRAYS_SUPPORTED, 2, 3, 5, 7, 8

MPI_ALLOC_MEM, 3, 6
MPI_ALLOC_MEM_CPTR, 6
MPI_COMM_CREATE_KEYVAL, 2-7
MPI_COMM_DUP_FN, 2, 3, 5, 7
MPI_COMM_F2C, 14
MPI_COMM_RANK, 3, 10, 11
MPI_COMM_RANK_F08, 10, 11
MPI_F_SYNC_REG, 3-7
MPI_FREE_MEM, 3
MPI_GET_ADDRESS, 3-7
MPI_Irecv, 12, 13
MPI_Irecv_cdesc, 12
MPI_Irecv_f08, 13
MPI_Irecv_f08ts, 12
MPI_Irecv_fts, 12
MPI_ISEND, 1-7, 9
MPI_ISEND_F08, 7
MPI_ISEND_F08TS, 7
MPI_ISEND_FTS, 7
MPI_OP_CREATE, 2-7
MPI_RECV, 7
MPI_RECV_F08, 7
MPI_REGISTER_DATAREP, 2-7
MPI_SEND, 3-7
MPI_SEND_F08, 7
MPI_TEST, 3, 5
MPI_WIN_ALLOCATE, 3
MPI_WIN_ALLOCATE_SHARED, 3
MPI_WIN_SHARED_QUERY, 3

PMPI_..._c2f, 15
PMPI_..._cdesc, 4
PMPI_..._f2c, 15
PMPI_SEND, 1