MPI-3.0 Fortran Tickets

(MPI Forum Meeting May 2011)

Rolf Rabenseifner, Jeff Squyres, Craig Rasmussen

Major ideas

- Solving the argument checking problems
  - Allowing also checking of wrong handle types
- Being backward compatible
- Allowing triplet array subscripts a(lb:ub:incr) everywhere,
  i.e., also in nonblocking routines
  - No need of MPI_Type_create_subarray for buffer descriptions.
  - No local copying
    - Prohibited on compiler level
    - Not required inside the MPI library
    - Although a quick MPI implementation can do such copying, as done currently by the compilers
- Additional features:
  - OPTIONAL ierror,
  - “status ignore” through overloading (i.e., also “*optional*”)
  - INTENT (IN, OUT, INOUT)

new = new since previous meeting

Major ideas, continued

- Fixing the Fortran-MPI-incompatibilities
  - At least with advices to users how to use Fortran in combination with MPI
  - Together with a chapter on "**Requirements on Fortran Compilers**" new
- New **mpi_f08** module with all features
  - Fully Fortran 2008 compatible definition of all MPI routines
  - Fortran 2003 work-around with nearly all compilers new
- Existing **mpi** module with several enhancements
- Different buffer handling in C-wrappers for
  - Fortran 2008 compilers
  - Older Fortran levels
- The use of mpif.h is strongly discouraged in the future
- In the future, i.e. all Fortran compilers support "assumed-type&rank" and BIND(C) CHARACTER*(*), it is possible to have only one set of C-wrappers for all three: mpi_f08 & mpi module, and mpif.h Still open question
- We should install a common source code infrastructure for new mpi_f08

Problems in discussion with Fortran committees:

- The Fortran 2008 method assumed-type & assumed-rank is now **perfect** new
  - TYPE(*), DIMENSION(..) buf
- Requirement:
  - Existing interface
    - CALL MPI_SEND(any possible actual argument)
    - SUBROUTINE MPI_SEND defined as implicit interface
  - New explicit interface must allow all possible existing calls to MPI_SEND, i.e., all possible actual buffer arguments
  - Should work, at least with BIND(C) for the choice buffer arguments
- Hard Problems:
  - Asynchronous new (i.e. nonblocking, one-sided, split-collective together with concurrent communication)
  - Handling of the new handles (based on Fortran sequence(?) derived types) new
- Fortran committees WG5 and J3 work on solutions → to be included into TR

#229-A – Overview on all related Fortran Tickets

- Ticket #229-A gives an overview on all related Fortran tickets
- Don’t worry that we have about 25 tickets
- They reflect separate decisions
- Before final voting,
  we can combine them to one single ticket
  (as done with nonblocking-collectives tickets)

- With all Fortran tickets together,
  we can solve all problems reported in MPI-2.2

- https://svn.mpi-forum.org/trac/mpi-forum-web/ticket/229

#230-B - New module "USE mpi_f08"

- New module mpi_f08
  - With all new features:
    - Full compile-time argument checking
- New wording (instead of Basic & Extended Fortran)
  - 3 Fortran support methods:
    - include ´mpif.h´
    - use mpi
    - use mpi_f08

Specifically for the MPI Forum, the Fortran 2008 standardization committee developed new syntax

TYPE(*), DIMENSION(..)

to define choice buffers in a standardized way

- Additional decision within #230-B: new
  - Callback prototypes via PROCEDURE(MPI_...) and ABSTRACT INTERFACE
  - Exceptions, i.e., old EXTERNAL (implicit) interface:
    - USER_FUNCTION for MPI_Op_create(user_fn, …)
    - DATAREP_CONVERSION_FUNCTION for MPI_Register_datarep(read_conversion_fn, write_conversion_fn, …)

Overview on all 3 Methods: **Include file mpif.h**

- Define all named MPI constants & declare MPI functions that return a value.
- For each MPI routine, an implementation can choose to use an implicit or explicit interface.
- The handles are defined as INTEGER
- mpif.h must be valid and equivalent for both fixed- and free- source form.
- **Advice to users: Instead of using mpif.h, the use of the mpi or mpi_f08 module is strongly encouraged. See … Reasons …**
  - **See Section 16.2.13, Advice to users** 

- Almost the same as MPI-2.2
- Only some small additions to MPI-2.2 (based on these tickets)
- Of course, all new stuff from other MPI-3.0 tickets and chapters will be added

Overview on all 3 Methods: **Module mpi**

- Define all named MPI constants & declare MPI functions that return a value.
- **Provide explicit interfaces for all MPI routines → compile-time argument checking.**
- The handles are defined as INTEGER
  - Same values as in mpif.h

An alternative will be in new mpi_f08 module

- An MPI implementation may provide in the mpi module other features that enhance the usability of MPI while maintaining adherence to the standard.
  For example, it may provide argument attributes INTENT(IN,OUT,INOUT) in these interface blocks.

- Backward compatible to MPI-2.2 for applications with bug-free syntax

- Details later

Overview on all 3 Methods: **Module mpi_f08**

- Define all named MPI constants & declare MPI functions that return a value.
- **Provide explicit interfaces for all MPI routines → compile-time argument checking.**
- **All handles are defined with named types.**
- **If the Fortran compiler provides *assumed type* and *assumed rank:***
  - **All choice buffers are declared with TYPE(*), DIMENSION(..)**
  - **MPI_SUBARRAYS = MPI_SUBARRAYS_SUPPORTED**

  → non-contiguous sub-arrays are also valid in nonblocking routines.

  Only if the target compiler does **not** support *assumed type* and *assumed rank* or an equivalent non-standard alternative:
  - Same requirements as for mpi module
  - MPI_SUBARRAYS = MPI_SUBARRAYS_NOT_SUPPORTED

  → non-contiguous sub-arrays are **not** valid also in nonblocking routines

```
Advice to users.
IF (MPI_SUBARRAYS ==
    MPI_SUBARRAYS_SUPPORTED)
   ...
ELSE IF (MPI_SUBARRAYS ==
    MPI_SUBARRAYS_NOT_SUPPORTED)
   ...
ELSE
   PRINT *, 'value not yet standardized'
ENDIF
```

- With this module, **new Fortran 2008 definitions are added for each MPI routine**, except for routines that are deprecated in MPI-2.2.
- **Each argument has an INTENT=IN, OUT, or INOUT attribute** if appropriate.
- **All ierror output arguments are declared as optional**, except for user-defined callback functions (e.g., comm_copy_attr_fn) and their predefined callbacks (e.g., MPI_NULL_COPY_FN).

Available with all 3 methods

- Each application routine can freely choose one of the 3 methods.
- **Compile-time constant MPI_SUBARRAYS is MPI_SUBARRAYS_SUPPORTED if all choice buffer arguments are implemented with TYPE(*), DIMENSION(..)**
  - Can be set different within all 3 methods
- MPI_SIZEOF, MPI_TYPE_MATCH_SIZE, MPI_TYPE_CREATE_F90_INTEGER, MPI_TYPE_CREATE_F90_REAL and MPI_TYPE_CREATE_F90_COMPLEX.
- **New MPI_F_SYNC_REG as one of several methods to solve register optimization problems.**
- Handle values are equivalent in *mpif.h* & *USE mpi*
- **Handles easily convertible to/from *USE mpi_f08***
- All new MPI-3.0 features are implemented in all 3 Fortran support methods

#231-C - Fortran argument checking with individual handles

- **This is the first major topic of new mpi_f08 module.**
- In mpi_f08, all handles use named types:

```
TYPE MPI_Comm
    SEQUENCE
    INTEGER   :: MPI_VAL
END TYPE MPI_Comm
```

Reason:
Otherwise handle not usable within an existing application's COMMON block

- Trivial conversion between old-style (module mpi) and new handles:
  comm_new%MPI_VAL = comm_old
- Same C-binding: one Fortran integer
  - The new handles do not require any changes for the wrappers written in C
- All new handle types are available in mpif_08, mpi, and mpif.h,

Yes	No	Abstain
?	?	?

new

  - But used in the routine definitions only in the mpi_f08 module
  - Helpful for conversion between old and new style Fortran application parts

Straw Votes:	Yes	No	Abstain
	11	-	5

#232-D: Existing module "USE mpi" with argument checking

- **Compile-time argument checking will now be mandatory for mpi module**
  - Can be done because most relevant Fortran compilers know directives like

    !DEC$ ATTRIBUTES NO_ARG_CHECK :: BUF

    !$PRAGMA IGNORE_TKR BUF

    REAL, DIMENSION(*) :: BUF
    - To be checked: Those directives imply same argument handling as with implicit interfaces in mpif.h for choice buffers
      → May need a Fortran complier-specific flag!
    - (such additional compiler flags are okay for compiling the module, but not a good idea for compiling the user application)
  - Only if the compiler has no such method, then overloaded functions are okay
    - See paper of M. Hennecke new
  - No cross-checking of different MPI handle types (because all are INTEGER)
    - **buf, count, datatype** ←→ **buf, datatype, count**

Straw Votes:	Yes	No	Abstain
	7	-	11

#233-E: The use of ‘mpif.h’ is strongly discouraged

- **Advice to users: Instead of using mpif.h, the use of the mpi or mpi_f08 module is strongly encouraged. See … Reasons …**
  - **See Section 16.2.13, Advice to users** new
  - Most have no compile-time argument checking
  - Too many bugs are in MPI applications, e.g., due to
    - Missing IERROR as last additional argument in Fortran
    - Status only as INTEGER,
      instead of INTEGER, DIMENSION(MPI_STATUS_SIZE)
    - Passing the wrong MPI handle types
    - …
  - Easy migration to USE mpi
    - For syntax-correct programs
- We do not deprecate mpif.h because
  - All new features should be still added to mpif.h
  - Because the same interface definition is used for mpif.h and USE mpi

Straw Votes:	Yes	No	Abstain
	9	-	10

#234-F: Choice buffers through "TYPE(*), DIMENSION(..)"

- **This is the second major topic of new mpi_f08 module.**
- **All choice buffers are declared with new Fortran 2008**
  - **TYPE(*), DIMENSION(..) = assumed type & assumed rank**
- Internally, a dope-vector (i.e., a descriptor) is passed to the Fortran wrapper (the dope-vector is generated by the complier and describes the buffer.)
- Based on this dope-vector, a virtual or real copying from non-contiguous to contiguous scratch arrays can be done.
- This contiguous scratch buffer is under control of MPI
- With non-blocking routines, it must not be released before MPI_Wait
- Implication:
  - All Fortran sub-arrays can be used also in non-blocking, one-sided, and split-collective MPI routines
  - Handling of sub-arrays can be done through Fortran syntax instead of MPI derived datatypes
- Exception: If compiler without TYPE(*), DIMENSION(..) → MPI_SUBARRAYS==MPI_SUBARRY_NOT_SUPPORTED

Many thanks to the Fortran 2008 standardization committee

How stable is a TR? Is it official?

"It is the intention of ISO/IEC JTC1/SC22/WG5 that the semantics and syntax specified by this technical report be included in the next revision of the Fortran International Standard without change unless experience in the implementation and use of this feature identifies errors that need to be corrected, or changes are needed to achieve proper integration, in which case every reasonable effort will be made to minimize the impact of such changes on existing implementations."

TR 29113, Draft N1845, page iv, Paragr. 7

Straw Votes:	Yes	No	Abstain
	13	-	2

Provided that TYPE(*), DIMENSION(..) will have Fortran Standard Quality

new

#235-G: Corrections to "Problems with Fortran Bindings" (MPI-2.2, p.481) and "Problems Due to Strong Typing" (p.482)

- No decisions, only new wording that is more correct.
- Corrections specific to the 3 methods (mpi_f08, mpi, mpif.h)

#236-H: Corrections to "Problems Due to Data Copying and Sequence Association" (MPI-2.2 page 482)

- No decision, mainly new wording to reflect the new methods and the constant MPI_F08_SUBARRAYS

```
USE mpi_f08
REAL :: buf(100)
CALL MPI_Irecv(buf(1:100:7), 15, MPI_REAL, …, req)
…
CALL MPI_Wait(req, …)
```

buf(1), buf(8), buf(15), buf(22), … buf(99)

Triplet-subscripts can now be used in non-blocking routines
if compile-time constant MPI_SUBARRAYS == MPI_SUBARRAYS_SUPPORTED

- New unsolved minor problem:
  Vector-subscripts still not usable in nonblocking calls, e.g. buf([1,2,24,25,33,34])
  - It is not planned to solve this problem, i.e, it is okay that only triplets are solved

#237-I: Corrections to "Problems Due to Fortran 90 Derived Types" (MPI-2.2 page 484)

- No decision, only new wording that correct this wrong section.
- Currently, MPI-2.2 says
  - "MPI does not explicitly support passing Fortran 90 derived types to dummy arguments. …
    Use of the SEQUENCE attribute may help here, somewhat.
- Reality is that MPI datatypes
  - work correctly with Fortran SEQUENCE and BIND(C) derived types, and
  - are not guaranteed to work for Fortran normal derived types.
- The section must be therefore corrected.

new

Must be checked

#238-J: Corrections to "Registers and Compiler Optimizations" (p. 371) and "A Problem with Register Optimization" (page 485)

- Additional advice about already known "Problems with MPI and Fortran optimization“ (or any future method)
  - New advice with the Fortran **TARGET** attribute
    - Solves problems with: nonblocking calls, MPI_BOTTOM, and 1sided
  - New advice with the Fortran **ASYNCHRONOUS** attribute
    - Solves problems with: nonblocking calls
  - Additional helper routine **MPI_F_SYNC_REG** to substitute the user-written DD(buf)
  - Module data and common blocks also work for all three problems
- New problem with "Temporary Memory Modifications"
  - Solved with **ASYNCHRONOUS**, but not with TARGET attribute, DD, MPI_SYNC_REG, module data, or common blocks
- **This is the third major topic of new mpi_f08 module: It is about correctness of MPI nonblocking, 1-sided, and MPI_BOTTOM based calls in Fortran.**

Had to be re-worked

Additional Problem:

buf1 + buf2 in one datatype

MPI_Send(buf1,1,datatype,…)

Straw Votes:	Yes	No	Abstain
	6	-	9

#238-J: Corrections to "Registers and Compiler Optimizations" (p. 371) and "A Problem with Register Optimization" (page 485)

- Three Optimization Problems:
  - Code movement and register optimization (was already discussed in MPI-2.0)
  - Temporary data movement (e.g., when using a GPU) new
  - Permanent data movement (e.g., as part of a garbage collection) new
- Four usage areas
  - Nonblocking MPI routines
  - One-sided MPI routines
  - Split-collective MPI routines
  - Usage of MPI_BOTTOM, or combining two variables through an MPI datatype

Optimization …	… may cause a problem when using:			
	Nonblocking	1-sided	Split-coll.	MPI_BOTTOM
Code movement and register optimization	YES	YES	no	YES
Temporary data movement	YES	YES	YES	no
Permanent data movement	YES	YES	YES	YES

#238-J: Corrections to "Registers and Compiler Optimizations" (MPI-2.2, p. 371) and "A Problem with Register Optimization" (page 485)

- **Solutions based on compromises:** new
  - **Minimize the burden for the application programmer**
  - **Minimize additional needs for the Fortran Standard**
  - **Minimize drawbacks on compiler optimizations**
  - **Minimize the requirements that are needed that MPI + Fortran guarantees correct execution of portable applications**

#238-J: Corrections to "Registers and Compiler Optimizations" (MPI-2.2, p. 371) and "A Problem with Register Optimization" (page 485)

Code movement and register optimization (was already discussed in MPI-2.0)

Optimization …	… may cause a problem when using:			
	Nonblocking	1-sided	Split-coll.	MPI_BOTTOM
Code movement and register optimization	YES	YES	no	YES

Solutions: — **Overhead may be**

- new **TARGET attribute** — **low-medium**
- new **Calling MPI_F_SYNC_REG** — **low**
  or a user defined routine (see DD in MPI-2.0) — **low**
- **Using module variables or COMMON blocks** — **low-medium**
- **VOLATILE** — **high-huge**

Wrong solution:

- new **ASYNCHRONOUS attribute** — **medium-high**

#238-J: Corrections to "Registers and Compiler Optimizations" (MPI-2.2, p. 371) and "A Problem with Register Optimization" (page 485)

Temporary data movement (e.g., when using a GPU)

new

Optimization ...	… may cause a problem when using:			
	Nonblocking	1-sided	Split-coll.	MPI_BOTTOM
Temporary data movement	YES	YES	YES	no

Overlapping communication and computation

Solutions:

- **None !!!!**

Alternative (***this is a hard restriction for the users !!!***): wow

- **Never use parts of a variable for communication / parallel I/O and another part for overlapping computation**

Wrong solution: **Overhead may be**

- **VOLATILE (*too expensive !!!*)** **high-huge**
- **ASYNCHRONOUS attribute (*does not work !!!*)** **medium-high**

#238-J: Corrections to "Registers and Compiler Optimizations" (MPI-2.2, p. 371) and "A Problem with Register Optimization" (page 485)

new

Permanent data movement (e.g., as part of a garbage collection)

Optimization ...	… may cause a problem when using:			
	Nonblocking	1-sided	Split-coll.	MPI_BOTTOM
Permanent data movement	YES	YES	YES	YES

Solutions:

- **None !!!!**

Okay?

Alternative (***this is a reasonable restriction for the implementors !!!***):

- **An MPI library + Fortran compiler is only MPI-3.0 compliant if this problem is solved!**

Wrong solution:

Overhead may be

- **VOLATILE (*too expensive !!!*)** — **high-huge**
- **ASYNCHRONOUS attribute (*does not work !!!*)** — **medium-high**

#239-K: IERROR optional

- In the current MPI Fortran interface, the IERROR dummy argument is mandatory.
- In the MPI C interface, the MPI routines can be called as
  - a function (i.e., the ierror value is returned), or
  - as a procedure (i.e., ignoring the ierror value),

  and therefore the ierror is optional.
- **With this ticket, the Fortran IERROR dummy argument is declared as OPTIONAL** in all MPI routines that provide an IERROR.
  - Exception: For user-defined callback functions (e.g., comm_copy_attr_fn) and their predefined callbacks (e.g., MPI_NULL_COPY_FN), ierror should not be optional
- An MPI implementation can also choose to use function overloading instead of implementing IERROR as optional
  - Advantage: Compile-time decision
  - Drawback: Doubling number of wrappers

- Implementation: We have to check that wrappers in C can test for this **optional** Fortran IERROR argument !!! → TR 29113 new

Straw Votes:	Yes	No	Abstain
	14	-	1

#240-L: New syntax used in the description of Fortran general interfaces

- New Fortran 95 style, e.g., INTEGER :: MPI_VERSION
- New wordings (currently none)

Straw Votes:	Yes	No	Abstain
	10	-	-

#241-M: Not including old deprecated routines

- Not to include deprecated routines into the new Fortran 2008 bindings.

Straw Votes:	Yes	No	Abstain
	Yes by acclamation		

#242-N: Arguments with INTENT=IN, OUT, INOUT

- **Use of INTENT=IN, OUT, INOUT attributes in all new Fortran 2008 bindings.**
- The Fortran attribute INTENT(IN) is used for all arguments that are IN arguments in the language-independent notation.
- For OUT or INOUT arguments in the language-independent notation, the Fortran attributes INTENT(OUT) or INTENT(INOUT) are used, with following exceptions:
  - If there exists a constant that can be provided as actual argument, then an INTENT attribute is not specified. Examples:
    - MPI_BOTTOM and MPI_IN_PLACE for buffer arguments;
    - MPI_UNWEIGHTED in sourceweights and destweights in MPI_Dist_graph_neighbors.
  - If the argument is a handle type argument and is implemented in C with call-by-value, then INTENT(IN) is specified. Examples:
    - All file-handles in MPI_Write routines;
    - the request in MPI_Grequest_complete.
    - Exception: MPI_Cancel with INTENT(IN) request
  - Buffers without INTENT
- An MPI implementation is allowed to use more restrictive INTENT
  - E.g., INTENT(IN) for send buffers
  - This is not explicitely mentioned, because optimization is allowed in general.

Exception from the exception:
- With USE mpi:no INTENT (will be corrected in next version after May 5, 2011)
- USE mpi_f08: INTENT(OUT) because "unweighted" is implemented by function overloading

new

Straw Votes:	Yes	No	Abstain
	11	-	3

#243-O: MPI_Status as a Fortran derived type

TYPE(MPI_Status) status

Reasons:

- The existing status(MPI_STATUS_SIZE) array is awkward
  - status%MPI_SOURCE versus status(MPI_SOURCE)
- Wrong accesses through integer indexes cannot be detected at compile-time
  - status(1) instead of status(MPI_SOURCE)
- Wrong arrays size is also not detected at compile-time
  - INTEGER status(1) instead of INTEGER status(MPI_STATUS_SIZE)

```
TYPE :: MPI_Status
 SEQUENCE
 INTEGER :: MPI_SOURCE
 INTEGER :: MPI_TAG
 INTEGER :: MPI_ERROR
 INTEGER :: MPI_internal_bytecnt_high ! implementation dependent
 INTEGER :: MPI_internal_bytecnt_low  ! implementation dependent
 INTEGER :: MPI_internal_cancal_flag  ! implementation dependent
END TYPE MPI_Status
```

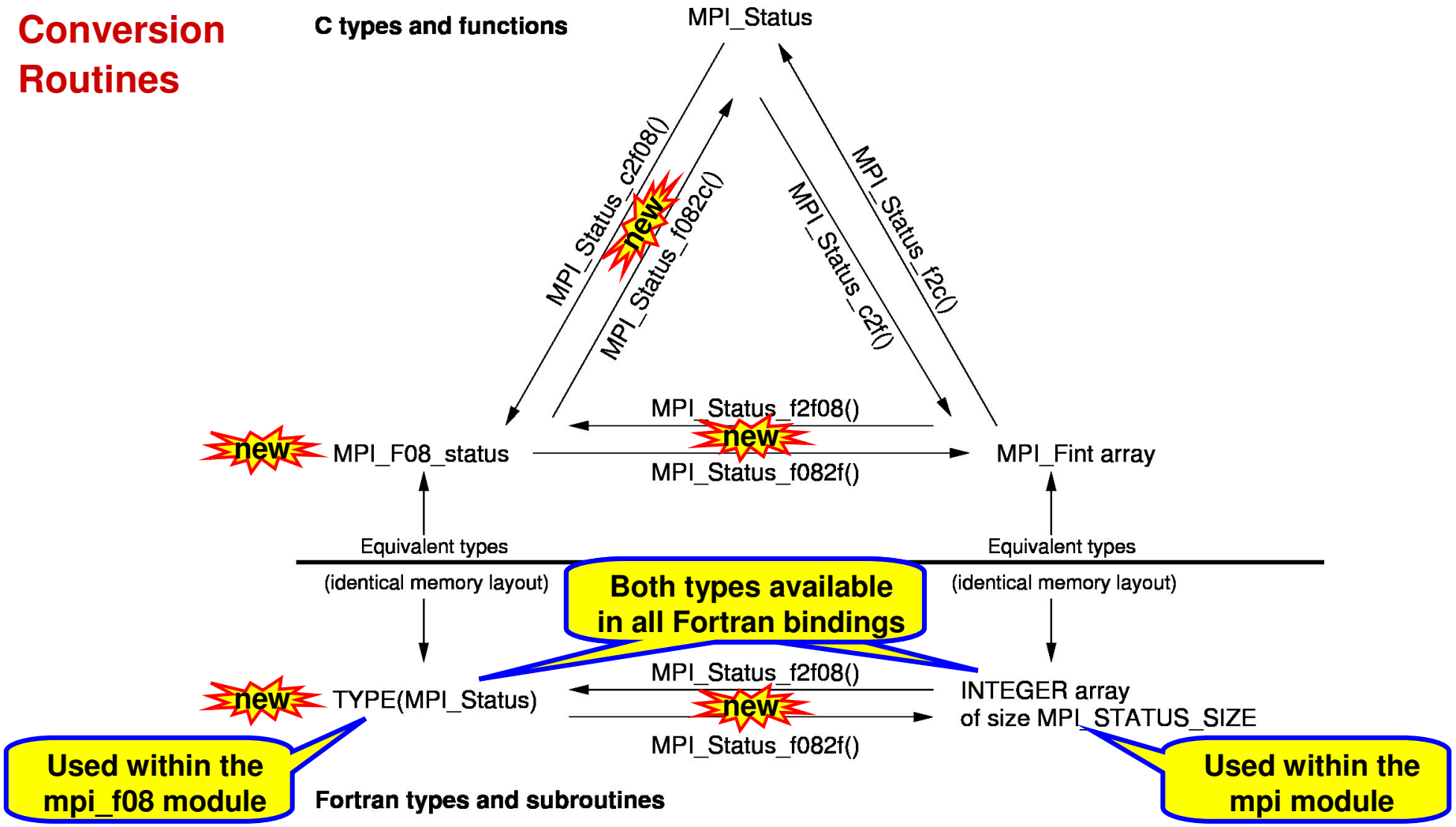
new

Reason:
Otherwise not usable within an existing application's COMMON block

Straw Votes:	Yes	No	Abstain
	9	-	5

#243-O: MPI_Status as a Fortran derived type

Conversion Routines



#244-P: MPI_STATUS(ES)_IGNORE with function overloading

With USE mpi_f08, the user can freely choose

- CALL MPI_Recv(buf,cnt,datatype,src,tag,comm,status,ierror)
- CALL MPI_Recv(buf,cnt,datatype,src,tag,comm, ierror)
- CALL MPI_Recv(buf,cnt,datatype,src,tag,comm,status)
- CALL MPI_Recv(buf,cnt,datatype,src,tag,comm)

- Some routines are often in the critical path:
  - Function overloading is at compile-time

  → no conditional branch at run-time

  → Function overloading is more efficient
- Only 36 routines with status output argument
- Same API cannot be done with OPTIONAL status argument, i.e., with OPTIONAL status, users must write
  - CALL MPI_File_write(fh,buf,count,datatype, IERROR=ierror)

  instead of
  - CALL MPI_File_write(fh,buf,count,datatype, ierror)
- Also MPI_ERRCODES_IGNORE and MPI_UNWEIGHTED

Note that here, ierror may be needed, because in all I/O routines, ERORS_RETURN is the default!

new

Same decisions as in C++

Straw Votes:	Yes	No	Abstain
	5	**2**	5

#246-Q: MPI_ALLOC_MEM and Fortran

- How to use MPI_ALLOC_MEM together with C-Pointers in Fortran. (instead of non-standard Cray-Pointers)

new

Also available in the mpi module, not in mpif.h

```
SUBROUTINE MPI_Alloc_mem(size, info, baseptr, ierror)
  USE, INTRINSIC :: ISO_C_BINDING
  INTEGER(KIND=MPI_ADDRESS_KIND), INTENT(IN) :: size
  TYPE(MPI_Info), INTENT(IN) :: info
  TYPE(C_PTR), INTENT(OUT) :: baseptr ! overloaded with the following…
  INTEGER(KIND=MPI_ADDRESS_KIND), INTENT(OUT) :: baseptr ! …type
  INTEGER, OPTIONAL, INTENT(OUT) :: ierror
END
```

new

```
SUBROUTINE MPI_Free_mem(base, ierror)
  TYPE(*), DIEMSION(..) :: base
  INTEGER, OPTIONAL, INTENT(OUT) :: ierror
END
```

- New Example 8.1:

new

```
USE mpi_f08   !   or USE mpi   (not guaranteed with INCLUDE 'mpif.h')
USE, INTRINSIC :: ISO_C_BINDING
TYPE(C_PTR) :: p
REAL, DIMENSION(:,:), POINTER :: a                ! no memory is allocated
INTEGER, DIMENSION(2) :: shape
INTEGER(KIND=MPI_ADDRESS_KIND) :: size
shape = (/100,100/)
size = 4 * shape(1) * shape(2)                    ! assuming 4 bytes per REAL
CALL MPI_Alloc_mem(size,MPI_INFO_NULL,p,ierr)     ! memory is allocated and
CALL C_F_POINTER(p, a, shape)                     ! now accessible through a
...
A(3,5) = 2.71;
...
CALL MPI_Free_mem(a, ierr)                        ! memory is freed
```

- New interface that can be used together with ALLOCATABLE arrays

Not done

Thanks to Dieter an Mey, who gave me an example in Feb. 2004

#245-R: Upper and lower case letters in new Fortran bindings

- The new interfaces in mpi_f08 look like:

```
SUBROUTINE MPI_Recv(buf, count, datatype, source, tag, comm, status, ierror)
        TYPE(*), DIMENSION(..) :: buf
        TYPE(MPI_Datatype), INTENT(IN)  :: datatype
        TYPE(MPI_Comm), INTENT(IN)  :: comm
        INTEGER, INTENT(IN)  :: count, source, tag
        TYPE(MPI_Status), INTENT(OUT) :: status
        INTEGER, OPTIONAL, INTENT(OUT) :: ierror
END
```

- Bill and Adam “hate" this → We will discuss the reasons

Straw Votes:	Yes	No	Abstain
	--- deferred ---		

#247-S: All new Fortran 2008 bindings – Part 1

- This ticket shows the principles and some special details

#248-T: All new Fortran 2008 bindings – Part 2

Half-automatically generated from the existing Fortran bindings!

A.4 Fortran 2008 Bindings with module mpi_f08
A.4.1 Point-to-Point Communication Fortran Bindings

SUBROUTINE MPI_Bsend(buf, count, datatype, dest, tag, comm, ierror) BIND(C)
TYPE(*), DIMENSION(..) :: buf
INTEGER, INTENT(IN) :: count, dest, tag
TYPE(MPI_Datatype), INTENT(IN) :: datatype
TYPE(MPI_Comm), INTENT(IN) :: comm
INTEGER, OPTIONAL, INTENT(OUT) :: ierror
END

SUBROUTINE MPI_Bsend_init(buf, count, datatype, dest, tag, comm, request, ierror) BIND(C)
TYPE(*), DIMENSION(..) :: buf
INTEGER, INTENT(IN) :: count, dest, tag
TYPE(MPI_Datatype), INTENT(IN) :: datatype
TYPE(MPI_Comm), INTENT(IN) :: comm
TYPE(MPI_Request), INTENT(OUT) :: request
INTEGER, OPTIONAL, INTENT(OUT) :: ierror
END
...

#250-V: Minor Corrections in Fortran Interfaces

- Typo in the existing Fortran Interface of MPI_INTERCOMM_MERGE:
  - INTRACOMM → **NEW**INTRACOMM
- Remove double definition of `request` in the Fortran binding type declaration part of MPI_SEND_INIT and MPI_BSEND_INIT
- Callback function prototypes in Fortran with same names as in C 

  - COMM/WIN/TYPE_CODY/DELETE/_ATTR_FN → …ATTR_**FUNCTION**
  - Same decision as in C and with other callback prototypes in Fortran.
  - The argument names are kept …_FN, as in C.

- Regular errata – in all three Fortran support methods
  - INOUBUF → INOU**T**BUF

Straw Votes: Yes No Abstain
Yes by acclamation

#252-W: Substituting dummy argument name "type" by "datatype" or "oldtype"

- To mimimize conflicts with language keywords (TYPE in Fortran), the dummy argument name "type" is substituted by "datatype" or "oldtype".
- "function" → user_fn, comm_errhandler_fn, …, handler_fn in MPI_OP_CREATE, MPI_COMM_CREATE_ERRHANDLER, …, MPI_ERRHANDLER_CREATE

- Note, with explicit interfaces, the user can freely choose between
  - Positional argument lists
    - CALL MPI_Send(buf,cnt,datatype,src,tag,comm,ierr)
  - Keyword-based argument lists and mixed lists
    - CALL MPI_Send(buf,cnt,datatype,source=src, &
& tag=13,comm=MPI_COMM_WORLD,ierror=ierr)

→ Dummy argument names *should be done correctly* & *should make sense*

Straw Votes: Yes No Abstain
Yes by acclamation

Implementation

- Craig Rasmussen is currently implementing a set of **Fortran-written** wrappers
  - From Fortran mpi_f08 with TKR_IGNORE Fortran 2003 work-around
  - To the C bindings

Section 16.2.16 Requirements on Fortran Compilers

- The compliance to MPI-3.0 (and later) Fortran bindings is not only a property of the MPI library itself, but is always a property of an MPI library together with the Fortran compiler it is compiled for.
  - *Advice to users.* Many MPI libraries are shipped together with special compilation scripts (e.g., mpif90, mpicc). These scripts start the compiler probably together with special options to guarantee this compliance. *(End of advice to users.)*
- An MPI library is only compliant with MPI-3.0 (and later), as referred by MPI_GET_VERSION, if all the solutions described in Sections 16.2.3 to 16.2.11 work correctly.

Summary on such requirements (slide 1) New

- Assumed-type and assumed-rank from Fortran 2008 TR 29113 is available;
  - Otherwise preliminary MPI-3.0 library with Fortran 2003 work-arround.
- Simply contiguous arrays and scalars must be passed to choice buffer dummy arguments with call by reference.
- SEQUENCE and BIND(C) derived types are valid as actual arguments passed to choice buffer dummy arguments and they are passed with call by reference. Derived type buffers
- The TARGET attribute solves code movement problems.
- Separately compiled empty Fortran routines with implicit interfaces and separately compiled empty C routines with BIND(C) Fortran interfaces (as MPI_F_SYNC_REG and user-written DD) solve code movement problems.
- The problems with temporary data movement are solved as long as the application uses different sets of variables for the nonblocking communication and the computation when overlapping communication and computation.
- Problems caused by automatic and permanent data movement (e.g., within a garbage collection) are resolved without any further requirements on the application program, neither on the usage of the buffers, nor on the declaration of application routines that are involved in calling MPI operations.

Rules about Correct-ness

Summary on such requirements (slide 2) New

- All actual arguments that are allowed for a dummy argument in an implicitly defined and separately compiled Fortran routine with the given compiler (e.g., CHARACTER(LEN=*) strings and array of strings) must also be valid for choice buffer dummy arguments with all Fortran support methods.
- The handle and status types in mpi_f08 (i.e., sequence derived types with INTEGER elements) are (handle) or can be (status) identical to one numerical storage unit or a sequence of those. These types must be valid at every location where an INTEGER and a fixed-size array of INTEGERs (i.e., handle and status in the mpi module and mpif.h) is valid, especially also within BIND(C) derived types defined by the application.
  - *Rationale.* This is not yet part of the draft N1845 of TR 29113 [36], but may be part of the nal version of this TR 29113 [35]. It is already implemented in some of the available Fortran compilers (e.g., ifort and pgi).
- Further requirements apply if the MPI library internally uses BIND(C) routine interfaces.

Major rules about back-ward compa-tibility

new

Open questions – … for users okay? … for the implementors okay? … and technically okay?

- Is the decision “sequence derived types for handles and status” okay?
- It is not expected that our new handles can be used officially in BIND(C) interfaces.
  - Some compilers already allow this.
  - Implication: Portable wrapper must be written in Fortran
  - Wrapper in C are not fully portable (still need to be adopted to the Fortran compiler)

  Is this okay?
- Is the decision “explicit callback prototypes for buffer-free routines” okay?
- Is the decision “implicit callback prototypes for routines with buffers” okay?
- Is the “wording about derived type user buffers and MPI_Type-create-struct” okay?
- Are the “solutions about code movement” together with the “requirements” okay?
- Is the restrictive solution for “temporary data movement” okay?
- With “permanent data movement”, is it okay to put the burden on the implementors?
- Are there link-time optimizations that still can produce wrong execution?

Problem with MPI_ALLOC_MEM

- Application 1 – using the a standard Fortran pointer TYPE(C_PTR) with MPI-2.2
  - Calls MPI_ALLOC_MEM that has an implicit interface (maybe within **mpif.h** or the **mpi** module)
  - This user has ignored the Example 8.1 because it uses a non-standard pointer
- Application 2 – Using Cray-Pointer together with a Fortran 95
  - Calls MPI_ALLOC_MEM with an implicit interface
  - Or having a compiler that maps Cray-Pointer with INTEGER(KIND=MPI_ADDRESS_KIND)
- With Fortran mpif.h, only the INTEGER(KIND=MPI_ADDRESS_KIND) BASEPTR is required. With the mpi module, a second, overloaded subroutine is required if the Fortran compiler supports ISO_C_BINDING:

```
MPI_ALLOC_MEM(SIZE, INFO, BASEPTR, IERROR)
  USE, INTRINSIC :: ISO_C_BINDING
  INTEGER :: INFO, IERROR
  INTEGER(KIND=MPI_ADDRESS_KIND) :: SIZE
  TYPE(C_PTR) :: BASEPTR
```

Conclusion

- Although separated into many tickets, it is one big packet.
- We needed 3 years to detect most of the MPI-Fortran-problems (1994-1997)
- We needed additional 13 year to hopefully solve them all (1997-2010) !
- And we still detect new ones ☺ ☹ ☺ ☹ ☺

- Thanks to the Fortran Standardization Committees WG5 and J3
  for their working together to solve the MPI-Fortran incompatibility problems.

Appendix

- List of all Fortran tickets

Tickets related to new MPI-3.0 Fortran Interface

- #229-A - Overview over all related tickets
- #230-B - New module "USE mpi_f08"
- #231-C - Fortran argument checking with individual handles
- #232-D - Existing module "USE mpi" with argument checking
- #233-E - The use of ‘mpif.h’ is strongly discouraged
- #234-F - Choice buffers through TYPE(*) DIMENSION(..) declarations
- #235-G - Corrections to "Problems with Fortran Bindings" (p.481) & "Strong Typing" (482)
- #236-H - Corrections to "Problems Due to Data Copying and Sequence Association" (482)
- #237-I - Corrections to problems due to "Fortran 90 Derived Types" (MPI-2.2 page 484)
- #238-J - Corrections to "A Problem with Register Optimization" (pages 371 and 485)
- #239-K - IERROR optional
- #240-L - New syntax used in all three (mpif.h, mpi, mpi_f08)

https://svn.mpi-forum.org/trac/mpi-forum-web/ticket/229 .. 253

Tickets related to new MPI-3.0 Fortran Interface

- #241-M - Not including old deprecated routines from MPI-2.0 - MPI-2.2
- #242-N - Arguments with INTENT=IN, OUT, INOUT
- #243-O - MPI_Status as a Fortran derived type
- #244-P - MPI_STATUS(ES)_IGNORE with function overloading
- #246-Q - MPI_ALLOC_MEM and Fortran
- #245-R - Upper and lower case letters in new Fortran bindings
- #247-S - All new Fortran 2008 bindings - Part 1
- #248-T - All new Fortran 2008 bindings - Part 2
- #249-U - Alternative formulation for Section 16.2 Fortran Support
- #250-V - Minor Corrections in Fortran Interfaces
- #252-W - Substituting dummy argument name "type" by "datatype" or "oldtype“, etc.
- #253-X - mpi_f08 Interfaces for new MPI-3.0 routines (not yet done)

- #251 is currently a helper ticket: Printable version of all tickets together