A Case for New MPI Fortran Bindings

C. E. Rasmussen[1], J. M. Squyres[2]

Advanced Computing Laboratory, Los Alamos National Lab
crasmussen@lanl.gov
Open Systems Laboratory, Indiana University
jsquyres@open-mpi.org

Abstract. The Fortran language has evolved substantially from the Fortran 77 bindings defined in the MPI-1 (Message Passing Interface) standard. Fortran 90 introduced interface blocks; subsequently, the MPI-2 standard defined Fortran 90 bindings with explicit Fortran interfaces to MPI routines. In this paper, we describe the Open MPI implementation of these two sets of Fortran bindings and point out particular issues related to them. In particular, we note that strong typing of the Fortran 90 MPI interfaces with user-choice buffers leads to an explosion of interface declarations; each choice buffer must be expanded to all possible combinations of Fortran type, kind, and array dimension. Because of this (and other reasons outlined in this paper), we propose a new set of Fortran MPI bindings that uses the intrinsic ISO_C_BINDING module in Fortran 2003. These new bindings will allow MPI interfaces to be defined in Fortran that directly invoke their corresponding MPI C implementation routines – no additional layer of software to marshall parameters between Fortran and C is required.

1 Introduction

The MPI-1 (Message Passing Interface) standard [5] has been very successful, in part, because it provided MPI bindings in both C and Fortran. Thus, programmers were able to write parallel message passing applications in the language of their choice. Most implementations of MPI are written in C (or C++) and provide a thin translation layer to effect the Fortran bindings.

The MPI-2 standard [4] continued this successful treatment of language interoperability by tracking the Fortran standard as it evolved by defining Fortran 90 bindings using *explicit* interfaces. Similar to the benefits enjoyed by C and C++ programmers, these new Fortran bindings allow the Fortran compiler to fail to compile a program if actual procedure arguments do not conform to the dummy arguments specified by the standard. This level of type safety (at the procedure call) is not possible with the original *implicit* Fortran 77 bindings.

Section 2 provides a brief overview of common implementation techniques and problems associated with the Fortran 77 bindings. Section 3 discusses the Open MPI [3] approach to implementing the Fortran 90 MPI bindings. It also includes details of automatic code generation techniques as well as practical problems that arise from the Fortran 90 MPI bindings specification. In particular,

the strong typing of the Fortran 90 explicit interfaces require the specification of a separate interface for each potential type, kind, and array dimension that could be specified by a user for an MPI user-choice buffer argument. This interface explosion for generic MPI procedures with choice buffer arguments is unattractive and can lead to extremely long build times for the MPI library. In response to new advances in the Fortran language standard [2], and to specific problems with the existing Fortran bindings discussed in Section 3, new Fortran MPI bindings are proposed in Section 4.

2 Open MPI Fortran 77 Bindings

Before the 1990 Fortran standard was introduced, Fortran did not provide the ability to explicitly define interfaces describing external procedures (functions and subroutines). In Fortran 77, external interfaces must be *inferred* from the parameters provided in the call to the external procedure. Thus, while the MPI Fortran 77 bindings define standard Fortran interfaces for calling the MPI library, the Fortran compiler does not check to ensure that correct types are supplied to the MPI routines by the programmer.

This lack of type safety actually makes it easier for an MPI implementation to provide a layer of code bridging between user Fortran and an MPI C implementation. Many MPI routines take the address of a data buffer as a parameter and a count, representing the length of the buffer (e.g., MPI_SEND). Since the Fortran convention is to pass arguments by address, virtually any Fortran type can be supplied as the data buffer, including basics scalar types (e.g., real) and arrays of these types.

In most MPI implementations, the Fortran bindings are a thin translation layer that marshals parameters between Fortran and C and invokes corresponding back-end C MPI functions. For scalar types, this only requires dereferencing pointers from Fortran before passing to the C implementation routines; array-valued parameters may be passed directly. MPI handle parameters must also be converted (typically either by pointer dereferencing or table lookup) to the back-end C MPI objects.

2.1 Issues

The primary difficulty in developing MPI Fortran 77 bindings is that Fortran does not define a standard for compiler generated symbols, For example, the symbol for MPI_SEND may be `MPI_SEND` or `mpi_send`, followed by one or two underscores (it will likely not be the C symbol `MPI_Send`). However this uncertainty is relatively easy to overcome and various strategies have evolved over time.

While type safety is still an issue (a programmer may mistakenly supply a real type for a buffer count parameter, for example), the Fortran 77 bindings have been successfully used in practice for many years. However, it should be noted that these bindings are implemented outside of the Fortran language specification

and may fail in future compiler versions. For example, an MPI subroutine with choice arguments may be called with different arguments types. This violates the letter of the Fortran standard, although such a violation is common practice [5].

3 Open MPI Fortran 90 Bindings

Several enhancements were made to Fortran in the 1990 standard. MPI-2 defined a Fortran 90 module and support for additional Fortran intrinsic numeric types. MPI interfaces could therefore both be defined for and expressed in Fortran. High-quality MPI implementations are encouraged to provide strong type checking in the MPI module, allowing the compiler to enforce consistency between the parameters supplied by the programmer and those defined in the MPI standard.

While this enhances type safety, there is no Fortran equivalent of the C `(void *)` data type used by the MPI C standard to declare a generic data buffer. *Every* data type that could conceivably be used as a data buffer must be declared in an *explicit* Fortran interface. The only fallback, for instance for user-defined data types, is for the programmer to resort to the older Fortran 77 *implicit* interfaces.

Not only must interfaces be defined for arrays of each intrinsic data type, but for each array dimension as well. Depending on the compiler, there may be approximately 15 type / size combinations.[1] Each of these combinations can be paired with up to a maximum of seven array dimensions. With approximately 50 MPI functions that have one choice buffer, this means that 5,250 interface declarations must be specified (i.e., 15 types $\times$ 7 dimensions $\times$ 50 functions). Note that this does not include the approximately 25 MPI functions with two choice buffers. This leads to an additional 6.8M interface declarations (i.e., $(15 \times 7 \times 25)^2$). Currently, no Fortran 90 compilers can compile a module with this many interface functions.

3.1 Code Generation

Because of the large number of separate interfaces that the MPI standard requires, automatic generation of this code is an attractive option. Chasm [6] was used to accomplish this task.

Chasm is a toolkit providing language interoperability between Fortran 90 and C / C++. It uses static analysis to produce a translation layer between language pairs by first parsing source files to produce an XML representation of existing interfaces and then using XSLT stylesheets to generate the final bridging code. Fig. 1 depicts the code generation process.

There are several different types of files generated by the Chasm XSLT stylesheets. The primary file is the MPI module declaring explicit Fortran interfaces for each MPI function. Similar to a C header file, this file allows the Fortran

[1] Assuming the compiler supports `CHARACTER`, `LOGICAL`{`1,2,4,8`}, `INTEGER`{`1,2,4,8`}, `REAL`{`4,8,16`}, and `COMPLEX`{`8,16,32`}.

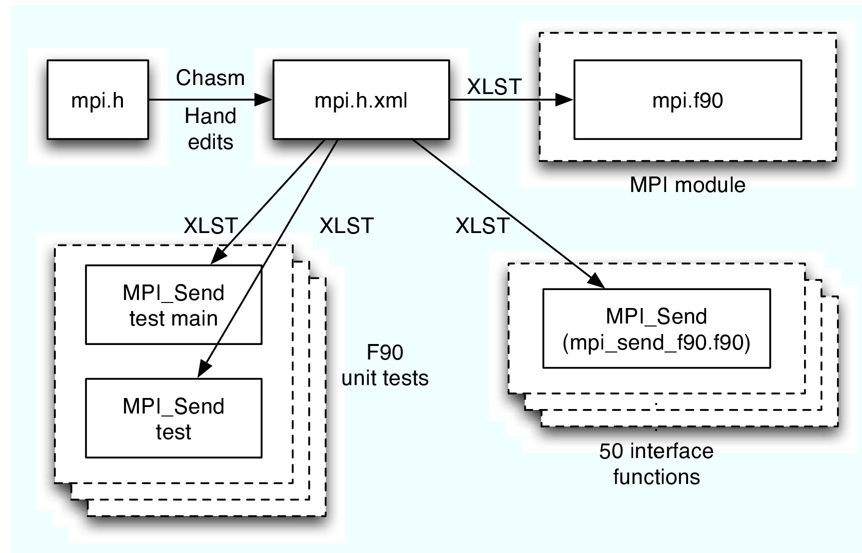

Fig. 1. Code generation for Fortran 90 MPI bindings in Open MPI.

compiler to check the actual parameter types supplied by user applications to make sure they conform to the interface. The actual Open MPI implementation of these interfaces is Fortran 77 binding layer (see Section 2). In addition, there are separate files generated to test each MPI function.

MPI functions with choice parameters are handled somewhat differently. They require an additional translation layer to convert Fortran array-valued parameters to C pointers when invoking the corresponding Fortran 77 binding.

All of the XSLT stylesheets take the XML file `mpi.h.xml` as input. This file was created by the Chasm tools from the Open MPI `mpi.h` header file and subsequently altered by hand to add additional information. An example of the annotations make to `mpi.h.xml` was the addition of the name `ierr` to MPI functions returning an error parameter. This gave notice to the Chasm XSLT stylesheets to create an interface for a subroutine rather than a function, with the `ierr` return value as the last parameter to the procedure (Fortran intent(out)) as defined by the MPI Fortran bindings. Another example of the modifications to `mpi.h.xml` was the "choice" tag added to given to MPI choice `(void *)` arguments. This allowed the XSLT stylesheets to create explicit interfaces for each possible type provided by the programmer.

3.2 Issues

While the Fortran 90 MPI bindings allow explicit type checking, there are a number of issues with these bindings and with the Open MPI implementation. No explicit interfaces were created for MPI functions with multiple choice parameters (e.g., MPI_ALLREDUCE), because this would have exploded the type

system to unmanageable proportions. User application utilizing these functions access the Fortran 77 layer directly with no type checking.

4 Proposed MPI Fortran BIND(C) Interfaces

The Fortran 2003 standard [2] contains a welcome addition that vastly improves language interoperability between Fortran and C. These additions are summarized in this section and a new set of Fortran MPI bindings based on this standard is proposed.

4.1 Fortran 2003 C Interoperability Standard

The Fortran 2003 standard includes the ability to declare interfaces to C procedures within Fortran itself. This is done by declaring procedure interfaces as BIND(C) and employing only interoperable arguments. It allows C function names to be given explicitly and removes the mismatch between procedure symbols generated by the C and Fortran compilers. Fortran BIND(C) interoperable types include primitive types, derived types or C structures (if all attributes are interoperable), C pointers, and C function pointers. BIND(C) interfaces are callable from either Fortran or C and may be implemented in either language.

This standard greatly simplifies language interoperability because it places the burden on the Fortran compiler to marshall procedure arguments and to create interoperable symbols for the linker, rather than placing the burden on the programmer. This includes the ability to use pass arguments by value. For MPI, this means that MPI C functions may be called directly from Fortran rather than from an intermediate layer. No additional work is needed other than to declare Fortran interfaces to the MPI C functions.

4.2 MPI C Type Mappings

The intrinsic ISO_C_BINDING module in the Fortran 2003 standard provides mappings between Fortran and C types. This mapping includes Fortran equivalents for the C types commonly used in MPI functions. For example, C integers, null-terminated character strings, and function pointers are declared in Fortran as INTEGER(C_INT), CHARACTER(C_CHAR), and TYPE(C_FUNPTR), respectively.

Most importantly, the ISO_C_BINDING module defines a Fortran equivalent to MPI choice (void *) buffers (TYPE(C_PTR)). This directly solves the interface explosion problem. It also allows interfaces to be declared for MPI functions with multiple choice buffers. In addition, the ISO_C_BINDING module provides functions for converting between Fortran pointers (including pointers associated with arrays) and the C_PTR type.

4.3 MPI_Send Example

An example of the proposed BIND(C) interface for MPI_SEND is shown in Figure 2.

```
1  interface
2    function MPI_Send(buf, count, datatype, dest, tag, comm) &
3                      BIND(C, name='MPI_Send')
4      use, intrinsic :: ISO_C_BINDING
5      use MPI_C_BINDING
6      type(C_PTR), value, intent(in) :: buf
7      integer(C_INT), value, intent(in) :: count
8      integer(kind=MPI_HANDLE_KIND), value, intent(in) :: datatype
9      integer(C_INT), value, intent(in) :: dest
10     integer(C_INT), value, intent(in) :: tag
11     integer(kind=MPI_HANDLE_KIND), value, intent(in) :: comm
12     integer(C_INT) :: MPI_Send
13   end function MPI_Send
14 end interface
```

Fig. 2. BIND(C) interface declaration for MPI_SEND.

Note the explicit name attribute given to the BIND(C) declaration in line 3 of Fig. 2. This attribute instructs the Fortran compiler to create the equivalent C symbol of the provided name. The MPI_C_BINDING module name is proposed in line 5 to distinguish it from the Fortran 90 MPI module name. The value attribute used in lines 6-11 instructs the Fortran compiler to use pass-by-value semantics and means that no dereferencing of the arguments need be done on the C side.

The TYPE(C_PTR) declaration in line 6 is the Fortran BIND(C) equivalent of a C (void *) parameter. The usage of this generic C pointer declaration removes the interface explosion for the Fortran 90 MPI_SEND implementation, as described in the previous section. A C_PTR can be obtained from a Fortran scalar or array variable that has the TARGET attribute via the C_LOC() intrinsic function. The TARGET attribute must be used in Fortran to specify any variable to which a Fortran pointer may be associated.

The Fortran equivalent of MPI handle types are declared in lines 8 and 11. The MPI_HANDLE_KIND attribute must be defined by the MPI implementation and allows flexibility in specifying the size of an MPI handle. At this point it is uncertain if MPI handle types declared in this way will work across all MPI implementations without the need for extra marshalling by the MPI library. The form proposed here should be considered tentative until MPI implementors can consider the consequences of this choice.

While not shown here, there are also interoperable equivalents for null terminated C strings (CHARACTER(C_CHAR), DIMENSION(*)) and C function pointers (C_FUNPTR). In addition, variables defined in the scope of the MPI_C_BINDING module may interoperate with global C variables, further merging the Fortran bindings with the MPI C implementations.

4.4 Issues

Unlike the MPI Fortran 77 and 90 bindings, the proposed bindings describe language interoperability within the Fortran language. Therefore the proposed bindings are guaranteed to work by the Fortran compiler (and companion C compiler) and are not just *expected* to work for a particular Fortran compiler vendor and version. Users will be expected to do some parameter conversions themselves, as noted above in regards to the use of the `C_LOC` intrinsic function.

In addition, the Fortran 2003 standard is new and vendors are just coming out with `ISO_C_BINDING` module implementations. Therefore, a period of time will be needed before one can test the proposed features against existing MPI implementations.

5 Conclusions

Because of evolving Fortran language standards and limitations in the MPI Fortran 77 and 90 bindings, we have proposed a new set of Fortran MPI bindings based on the intrinsic `ISO_C_BINDING` module. These new bindings have several distinct advantages:

1. They solve the interface explosion problem of the Fortran 90 bindings through the use of `TYPE(C_PTR)`. This new type allows a direct mapping to and from the C `(void *)` choice buffers.
2. They allow direct calls to the MPI C implementation from Fortran. This is more efficient and is less error prone, as the MPI implementor does not need to maintain and test an extra binding layer. The Fortran compiler is responsible for marshalling between C and Fortran data types, not the MPI library.
3. The names of the C functions implementing the MPI procedures can be specified in Fortran. This means that the tricks required to create common symbols between compilers are no longer needed.
4. The proposed bindings are defined entirely within the Fortran language and are guaranteed to work by the Fortran compiler. The proposed bindings are not compiler dependent. While not likely, Fortran 77 bindings are implemented outside of the Fortran language specification and may fail in future compiler versions.

It should be pointed out that as of this writing, only two major compiler vendors support the Fortran 2003 `ISO_C_BINDING` module (others will likely do so by the fall of 2005 [1]). However, even this support is partial. Thus, there exists a window of opportunity to consider and modify the proposed bindings before widespread adoption.

To this end, we will post the full set of new Fortran bindings and a reference implementation on the Open MPI web site (`http://www.open-mpi.org/`) and solicit comments and feedback from the Fortran HPC community.

Acknowledgments

This work was supported by a grant from the Lilly Endowment and National Science Foundation grants EIA-0202048 and ANI-0330620.

Los Alamos National Laboratory is operated by the University of California for the National Nuclear Security Administration of the United States Department of Energy under contract W-7405-ENG-36.

References

1. Personal communication with compiler vendors. Meeting 168 of the J3 Fortran Standards Committee, August 2004.
2. Fortran 2003 Final Committee Draft, J3/03-007R2. see `www.j3-fortran.org`.
3. E. Garbriel, G.E. Fagg, G. Bosilica, T. Angskun, J. J. Dongarra J.M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R.H. Castain, D.J. Daniel, R.L. Graham, and T.S. Woodall. Open mpi: Goals, concept, and design of a next generation mpi implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, 2004.
4. A. Geist, W. Gropp, S. Huss-Lederman, A. Lumsdaine, E. Lusk, W. Saphir, T. Skjellum, and M. Snir. MPI-2: Extending the Message-Passing Interface. In *Euro-Par '96 Parallel Processing*, pages 128–135. Springer Verlag, 1996.
5. Message Passing Interface Forum. MPI: A Message Passing Interface. In *Proc. of Supercomputing '93*, pages 878–883. IEEE Computer Society Press, November 1993.
6. Craig E Rasmussen, Matthew J. Sottile, Sameer Shende, and Allen D. Malony. Bridging the language gap in scientific computing: The Chasm approach. *Concurrency and Computation: Practice and Experience*, 2005.