

MPI

Application Binary Interface (ABI)

Standardization

Jeff Hammond
Principal Software Architect, NVIDIA
Co-chair, MPI Forum ABI WG
jeff.science@gmail.com

MPI ABI Status Quo

MPI is an **API** standard, which defines the source code behavior in C (C++) and Fortran. The **compiled** representation of MPI features is implementation-defined.

If you **compile** with one of the following MPI families, you **MUST run** with the same.

1. MPICH / Intel MPI / MVAPICH / Cray MPI
2. Open MPI / NVIDIA HPC-X / Amazon MPI / IBM Spectrum MPI

Family 1 exists because there was a demand for interoperability with Intel MPI due to the prevalence of usage in ISV codes.

Family 2 is not guaranteed to be consistent, especially across major versions.

API versus ABI

API

```
int MPI_Bcast(void * buffer, int count, MPI_Datatype d, int root, MPI_Comm c);
```

MPI_Datatype and MPI_Comm are unspecified types

ABI

```
typedef struct ompi_datatype_t * MPI_Datatype; // Open MPI family
```

```
typedef int MPI_Datatype; // MPICH family
```

Lots of other stuff like SO names, SO versioning, calling convention, etc.

Why?

Modern software use cases:

- Third-party **language** support, e.g. Python, Julia, Rust, etc.
- **Package** distribution, e.g. Spack, Apt, etc.
- **Tools** become implementation-agnostic
- **Containers**
- More efficient **testing** (build only once)

We can:

- Architectural reasons not to are gone
- Two platform ABIs cover >90% of HPC platforms

MPI Application Binary Interface Standardization

Jeff R. Hammond
NVIDIA Helsinki Oy
Helsinki, Finland

NVHPC SDK, Fortran

Marc Pérache
CEA, DAM, DIF
Arpajon, France

wi4mpi, containers, MPC

Gonzalo Brito Gadeschi
NVIDIA GmbH
Munich, Germany

Rust, containers

Lisandro Dalcin
Extreme Computing Research Center

KAUST

Thuwal, Saudi Arabia

dalcin@kaust.edu.sa
Python

Jean-Baptiste Besnard

ParaTools

Bruyères-le-Châtel, France

jbbs@paratools.org
TAU, E4S

Joseph Schuchart

University of Tennessee, Knoxville

Knoxville, Tennessee, USA

schuchart@utk.edu
Open MPI

Hui Zhou

Argonne National Laboratory

Lemont, Illinois, USA

zhou@anl.gov
MPICH

Erik Schnetter

Perimeter Institute for Theoretical
Physics

Waterloo, Ontario, Canada

esc@perimeterinstitute.ca
Julia, MPItrampoline

Jed Brown

University of Colorado Boulder

Boulder, Colorado, USA

jed@colorado.edu
PETSc, Rust

Simon Byrne

California Institute of Technology

Pasadena, California, USA

simonbyrne@caltech.edu
Julia

Design Decisions

The Status Object

```
typedef {  
    int MPI_SOURCE;  
    int MPI_TAG;  
    int MPI_ERROR;  
    int mpi_reserved[5];  
} MPI_Status;
```

Bigger than MPICH (5) and OMPI (6).

Reserves room for a 64b count, a 32b cancelled, and a 64b pointer, for example.

32 bytes is good for alignment.

Handles

```
typedef struct MPI_ABI_Comm * MPI_Comm;  
typedef struct MPI_ABI_Request * MPI_Request;  
...
```

Satisfies existing requirements (= comparison, fits into a pointer because attributes).

Supports type-safety. Compilers know that MPI_Comm is not MPI_Group.

Downside: conversions to/from Fortran are not free like MPICH (at least with LP64).

Handle Constants

```
0b 0000 0000 0000 to 0b 1111 1111 1111 reserved # zero page
0b 0000 0000 0000 invalid handle (detect uninitialized data)
0b 000* **** **** Everything except datatypes
0b 001* **** **** MPI_Datatype branch
0b 0010 **** **** Sufficient for all datatypes today
0b 0011 **** **** Reserved for future use
MPI_<handle>_NULL is always the handle prefix followed by 0s.
```

Handle Constants: Fixed-size datatypes

0b 0010 xxxxx yyy 5b for category, 3b for kind

00... not strictly fixed-size

01... C/C++ fixed-size

10... reserved

11... Fortran fixed-size

^^^ encoded size bits (log2 of size in bytes)

^ fixed-size bit

Implementations can test for fixed-size, then mask and shift to get the element size in bytes.

Handle Constants: C/C++ fixed-size kinds

0b000: MPI_INT(n)_T

0b001: MPI_UINT(n)_T

0b010: <float (n)b>

0b011: (size=1) ? MPI_CHAR : <C complex 2x(n/2)b>

0b100: (size=1) ? MPI_SIGNED_CHAR : reserved datatype

0b101: (size=1) ? MPI_UNSIGNED_CHAR : reserved datatype

0b110: (size=2) ? <C++ bfloat16_t> : reserved datatype

0b111: (size=1) ? MPI_BYTE : <C++ complex 2x(n/2)b>

Handle Constants: Fortran fixed-size kinds

0b000: MPI_INTEGER(n)

0b001: MPI_LOGICAL(n) (not standard)

0b010: MPI_REAL(n)

0b011: (size=1) ? MPI_CHARACTER : MPI_COMPLEX(n)

Handle Constants: Other datatypes

`MPI_INT`, `MPI_LONG`, even `MPI_FLOAT` are not fixed-size datatypes and require a size lookup.

It may save a few cycles to use `MPI_BYTE` and `sizeof()`, but measurements show no impact (~11 nanoseconds with both MPICH and OMPI).

`MPI_INTEGER`, `MPI_REAL` and `MPI_DOUBLE_PRECISION` are not fixed-size datatypes. More on this later...

Handle Constants - Table sizes

32-61 Op

256-288 Comm, Group, Win, File, Session, Message, Errhandler, Request

512-601 Datatype: variable-size and C/C++ fixed-size

602-623 with extras (e.g. `std::complex<__float128>`)

704-747 Datatype: Fortran fixed-size

Integer Constants

Requirements:

- Position sequences: 0..n (MPI_SUCCESS..MPI_ERR_LASTCODE)
- XOR-able, i.e., 2^k (e.g. MPI_MODE_NOCHECK)
- Negative (MPI_ANY_SOURCE)
- Sizes (e.g. MPI_BSEND_OVERHEAD)
- Ordered subsets (e.g. MPI_THREAD_*)
- Arbitrary (e.g. MPI_ORDER_FORTRAN)

Except for error codes, array sizes and XOR-ables, all integer constants are unique and negative. Error messages can tell user what they passed as it appears in the source code.

Other Constants

```
// Buffer Address Constants

#define MPI_BOTTOM          ((void*)0)
#define MPI_IN_PLACE       ((void*)1)

// Constants Specifying Empty or Ignored Input

#define MPI_ARGV_NULL      ((char**)0)
#define MPI_ARGVS_NULL    ((char***)0)
#define MPI_ERRCODES_IGNORE ((int*)0)
#define MPI_STATUS_IGNORE  ((MPI_Status*)0)
#define MPI_STATUSES_IGNORE ((MPI_Status**)0)
#define MPI_UNWEIGHTED     ((int*)2)
#define MPI_WEIGHTS_EMPTY  ((int*)3)
```


Special integer types

MPI integer types:

- MPI_Aint is intptr_t because that satisfies all of the requirements
 - Segmented addressing is irrelevant and should be removed.
 - Wide (128b) pointers (e.g. CHERI) are difficult to support with 64b addresses.
- MPI_Offset should be int64_t because that will be sufficient for ~30 years
 - We are still arguing about this, because apparently sparse files with 128b offsets are a thing.
- MPI_Count should be int64_t except on 128b systems (which we ignore)
 - Divorcing this from MPI_Offset has been discussed...
- MPI_Fint must match the Fortran compiler *and is thus impossible*
 - This exists in C via f2c/c2f as well as MPI_Type_size(MPI_INTEGER,..)

MPI ABI Packaging

- The header is `abi/mpi.h`
 - `#include <mpi.h>` still works - no code changes required to adopt ABI
 - The Forum should distribute a standard header for convenience
- The library is `libmpi_abi.ext`
 - Implementations are instructed to use platform-specific SO versioning conventions
 - The Forum should distribute a standard SO for convenience
- The ABI is versioned independently from the API
 - ABI starts with 1.0
 - Backwards-compatible changes (e.g. new handle type) increment the minor version
 - Backwards-incompatible changes increment the major version
 - Adding a new function to the API does not change the ABI

Implementing the standard ABI

1. **Standalone:** dlopen MPI, dlsym everything, translate everything at runtime.
 - wi4mpi (CEA)
 - MPItrampoline (Erik Schnetter)
 - Mukautuva (Jeff Hammond)
2. **Integrated:** the MPI library implements the ABI in a separate header+library and does all the conversions to the existing ABI internally.
 - MPICH has done this already
3. **Native:** the MPI library implements the ABI throughput.

MPI	Messages/second
Intel MPI 2021.9.0	4658939.64
+ Mukautuva	4606473.95
MPICH dev UCX [1]	13643117.42
+ Mukautuva	12278837.03
MPICH dev UCX ABI [2]	13643378.98

1. --enable-error-checking=no --enable-fast=Os --enable-g=none --with-device=ch4:ucx

2. Same as 1 plus --enable-mpi-abi

<https://github.com/jeffhammond/mukautuva>

When?

- Targeting MPI-Next as a single-feature ABI-only release (mid-2025?).
- Mukautuva, wi4mpi, and MPItrampoline can support this immediately.
- MPICH has a prototype already.
- Open MPI has not implemented this but they say it's easy.

Diffusion: upstream -> release -> packaging, etc.

FAQ

- Fortran is the only remaining issue. The C-specific part is done.
- Launchers are not part of the ABI. There are at least two options:
 - Slurm and PBS launchers are supported by all the major MPIs already.
 - mpirun can set the shared library to use, in which case the launcher and library will match.
- We will still use mpi.h for source compatibility. SO will be libmpi_abi.so.
- SO expected to follow platform-specific SO versioning convention.
- Wrapper scripts (e.g. mpicc) are not standard but we will probably have “mpicc_abi” or “mpicc -abi”.
- MPICH and Open MPI will continue to support their existing ABIs.

The End