Los Alamos NATIONAL LABORATORY

Moving MPI ABI ahead with Header Files

Generating C header files for Open MPI with Python

Joseph Downs

2025-08-07

LA-UR-25-28211

NNSA National Nuclear Security Administration

Managed by Triad National Security, LLC,for the U.S. Department of Energy's NNSA

Background

- Terms
- Why ABI?
- Handles

MPI Standard

- Binding Tool
- ABI JSON
- Issues

Open MPI

- Header Generation
- Internal Implementation
- Issues

Terms

- MPI – Message Passing Interface
- ABI – Application Binary Interface
- OMPI – Open MPI

Why ABI?

- Avoid recompilation/reconfiguration
- Better support in containerized environments
- Better support for modern language implementations of MPI (C++, Julia, Rust, etc.)
- Standardization of opaque handle format

Handles

- Handles are data structures used by the library to pass around data
- Predefined ABI-compliant handles are integers set in the standard

Binding Tool

- The standard makes use of Python to generate both LaTeX and JSON of function definitions
- JSON used in the separate library `pympistandard` to assist in generating C and Fortran code of MPI bindings
- LaTeX used to render the MPI Standard document

Binding Tool

Listing 1: Binding tool calls for `MPI_Send`

```
\begin{mpi-binding}
  function_name("MPI_Send")
  parameter("buf", "BUFFER", constant=True,
            desc=r"initial address of send buffer")
  parameter("count", "POLYXFER_NUM_ELEM_NNI",
            desc=r"number of elements in send buffer")
  parameter("datatype", "DATATYPE",
            desc=r"datatype of each send buffer element")
  parameter("dest", "RANK", desc=r"rank of destination")
  parameter("tag", "TAG", desc=r"message tag")
  parameter("comm", "COMMUNICATOR")
\end{mpi-binding}
```

Binding Tool

MPI_SEND(buf, count, datatype, dest, tag, comm)

IN	buf	initial address of send buffer (choice)
IN	count	number of elements in send buffer (nonnegative integer)
IN	datatype	datatype of each send buffer element (handle)
IN	dest	rank of destination (integer)
IN	tag	message tag (integer)
IN	comm	communicator (handle)

C binding

```
int MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest,
             int tag, MPI_Comm comm)

int MPI_Send_c(const void *buf, MPI_Count count, MPI_Datatype datatype,
             int dest, int tag, MPI_Comm comm)
```

Figure 1: Rendered binding LaTeX for `MPI_Send`

ABI JSON

- Using the binding tool as a model, I added additional code to the standard to generate a JSON for ABI constants and other information
- This is used in OMPI to generate `#define` and `enum` in header files

Listing 2: Snippet of the JSON

```
"mpi_success": {
 "handle_types": {
   "c": {
     "type": "int",
     "description": "[...]"
   },
   "f90": {
     "type": "INTEGER",
     "description": "\\ftype{INTEGER}"
   },
   "f08": {
     "type": "INTEGER",
     "description": "\\ftype{INTEGER}"
   }
 },
 "category": "ERROR_CLASSES",
 "macro": "\\error",
 "datatypes": {},
 "name": "MPI_SUCCESS",
 "abi_value": 0
},
```

Issues

- Missing definitions for `MPI_T_*` handles in the standard
- `T` stands for “tool” — used in tool interfaces

Header Generation

- I used Python, the ABI JSON, and `pympistandard` library to generate `#define`, function prototypes, and callback functions
- Generated two header files: one with `_ABI_INTERNAL` identifiers and another with standard ABI identifiers
- `_ABI_INTERNAL` identifiers used for the internal OMPI implementation to support ABI without altering existing logic

- Functions defined to convert from ABI to internal OMPI implementation

Listing 3: Internal OMPI ABI Implementation of `MPI_Send`

```
int MPI_Send(const void * buf, int count,
             MPI_Datatype_ABI_INTERNAL type,
             int dest, int tag, MPI_Comm_ABI_INTERNAL comm)
{
    int ret_value;
    MPI_Datatype type_tmp = ompi_convert_abi_datatype_intern_datatype(type);
    MPI_Comm comm_tmp = ompi_convert_abi_comm_intern_comm(comm);
    ret_value = ompi_abi_send(buf, count, type_tmp, dest, tag, comm_tmp);
    return ompi_convert_abi_error_intern_error(ret_value);
}
```

Issues

- I fixed some bugs in `pympistandard`:
  - Missing new functions and types released in MPI 5.0
  - Embiggened profiling functions not supported

Questions?

What's Next?

- ABI-related code merged into OMPI's `main` branch within the next few months
- Potentially available in OMPI v6.0 — no definite date on release yet

The Embiggening

- Many MPI functions use C `int` types for `count` parameters
- 4-byte `int` is not always enough; a new datatype, `MPI_Count` was created to allow for larger values
- For compatibility, functions have standard and “large count” variants (denoted by an appended `_c` to the function name)
- Such functions are considered “embiggened” functions