MPI Debugging with Handle Introspection

Brock-Nannestad, Laust; DelSignore, John; Squyres, Jeffrey M.; Karlsson, Sven ; Mohror, Kathryn

Publication date:
2014

Document Version
Publisher's PDF, also known as Version of record

MPI Debugging with Handle Introspection

Laust Brock-Nannestad*, John DelSignore†, Jeffrey M. Squyres‡, Sven Karlsson*, Kathryn Mohror§
*Technical University of Denmark
Email: {laub|svea}@dtu.dk
†Rogue Wave Software, Inc.
Email: John.DelSignore@roguewave.com
‡Cisco Systems, Inc.
Email: jsquyres@cisco.com
§Lawrence Livermore National Laboratory
Email: kathryn@llnl.gov

***Abstract*—The Message Passing Interface, MPI, is the standard programming model for high performance computing clusters. However, debugging applications on large scale clusters is difficult. The widely used *Message Queue Dumping* interface enables inspection of message queue state but there is no general interface for extracting information from MPI objects such as communicators. A developer can debug the MPI library as if it was part of the application, but this exposes an unneeded level of detail.**

The Tools Working Group in the MPI Forum has proposed a specification for *MPI Handle Introspection*. It defines a standard interface that lets debuggers extract information from MPI objects. Extracted information is then presented to the developer, in a human readable format. The interface is designed to be independent of MPI implementations and debuggers.

In this paper, we describe our support for introspection in the TotalView debugger and test it against a reference introspection implementation in Open MPI. We also describe how the debugger interfaces with the MPI implementation.

I. Introduction

Typically, developers possess the domain knowledge required to understand applications – from the control flow to the data structures employed. This knowledge is essential when debugging software errors, as it allows developers to understand the data presented by the debugger. Using MPI adds a layer of state that is not accessible through a conventional debugger: Part of the application's state will be stored within the MPI implementation, and in data structures specific to the implementation. This MPI state can be extracted by debugging the MPI library, but this is beyond the scope of many application developers.

In general, developers are not interested in internal MPI implementation details. Instead, they are interested in the MPI *API-level state*. This is the state of the application at the level of the MPI primitives being used. The task of debugging is simplified, if this abstract MPI state can be presented to the developer. MPI handle introspection aims to present MPI object data in an implementation-agnostic way and thereby simplify the debugging process.

Using the proposal from the Tools Working Group [1], we add introspection support to a development version of the TotalView debugger [2]. The necessary modifications to the MPI library, we implement in Open MPI [3]. We use our implementation to inspect MPI communicator state through TotalView and describe our challenges and experiences.

The rest of this paper is organized as follows: Section II discusses the issues developing a standardized interface between debuggers and MPI libraries. Section III gives an evaluation of the implementation in TotalView and Open MPI. Finally, section IV concludes.

II. The Introspection Interface

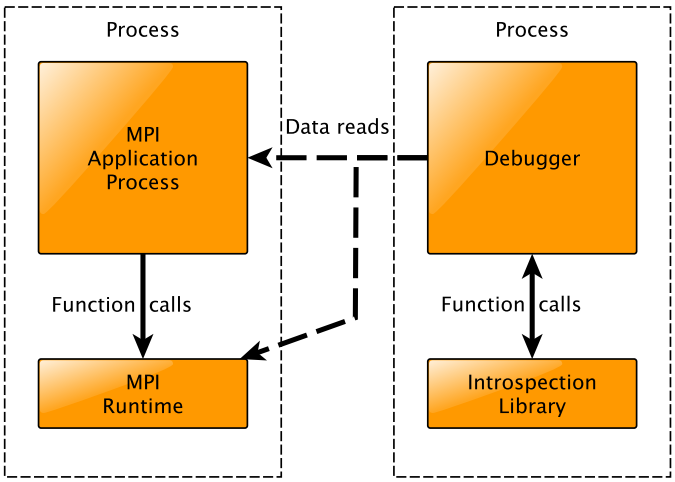

Fig. 1. The overall concept. The arrows indicate interactions between the different components. The debugger uses the MPI introspection library to decode information from the application. To do this, the introspection library uses the debugger to read raw data from the application. Note: Only one MPI process is shown.

Extending the capabilities of debuggers with plug-ins or helper libraries has already proven to be a reliable solution: The technique has been used for message queue dumping in MPI for 15 years [4]. Subsequently, a similar approach has been seen within OpenMP debugging [5], where such an interface is now being proposed for inclusion into the standard [6].

A. Responsibility of the Debugger

The purpose of a debugger is to allow safe inspection and manipulation of the application being debugged. Ideally,

the debugger provides mappings between raw memory and symbols, or data structures, in the source code.

Support for introspection requires minimal changes to the debugger. It must allow debugging of the application through a helper library. TotalView already supports dynamic libraries, as these are used by the previously mentioned OpenMP and MPI debugging facilities. An overall architecture diagram can be seen in figure 1: The debugger and the introspection library execute as a single process, while the MPI application uses the MPI runtime normally. The debugger and the introspection library call each other to request information, while only the debugger inspects the application.

B. Responsibility of the introspection library

Introspection support is implemented by the MPI vendor as a dynamic library. The library provides a set of functions for querying and decoding MPI information in MPI handles. These functions are called by the debugger. The proposed specification currently defines query functions for communicators, error handlers, request and status objects.

Figure 2 presents a sequence diagram for loading the dynamic library and setting up the *callback functions*, that the debugger passes on to the library for later use. Initially, the debugger detects that it is debugging an MPI program by the existence of a specific symbol in the *target process*. This symbol points to a list of locations for the dynamic library. Once the library is located, the debugger loads it. If successful, initialization begins. During initialization, the debugger performs basic sanity checking by ensuring that all the symbols defined by the interface can be resolved in the library. It reports an error if this is not the case.

After initial sanity checking, the debugger calls `mpidbg_init_once` to perform initialization and to pass on a set of callback functions. The library is running in the address space of the debugger and depends on callback functions in the debugger for resource management, I/O, and most importantly, access to the target process.

All accesses to the target process are performed by the debugger. This ensures that reads from invalid addresses are gracefully handled. Invalid reads are likely to occur if the library is passed an invalid MPI handle, due to a fault in the program being debugged. Additionally, this transparently allows debugging of live and dead processes, e.g. through core files.

C. Performing queries

Figure 3 shows the debugger querying a communicator handle. To begin with, the debugger obtains the address of the `MPI_Comm` handle in the target address space. It then calls `mpidbg_comm_query()` to set up the query. If the introspection library recognizes the address as a valid communicator handle, it fetches any relevant information – through the debugger – from the target process. The debugger is responsible for reading from the target process on behalf of the library. Once the library has fetched the required data, it returns an `mpidbg_comm_handle_t`. This handle is used

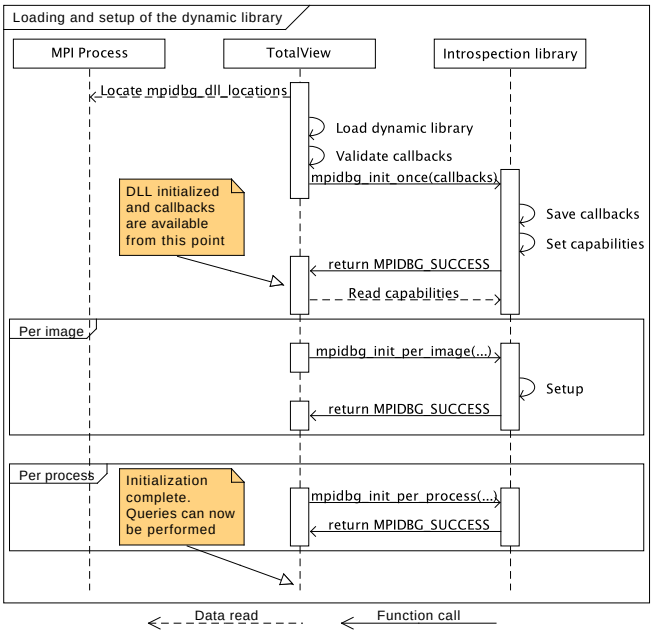

Fig. 2. Sequence diagram of TotalView loading the introspection dynamic library and the initial setup.

by the debugger for additional queries. In the case presented in figure 3, it calls `mpidbg_query_basic()`, returning basic information on the communicator such as rank, name, and type.

The two step approach lets the introspection library prefetch data regarding the communicator during `mpidbg_comm_query()` and cache it. It is also a way of allowing the debugger to inspect old state: the debugger can keep the handle, and perform new queries on it, even after the application has been resumed.

III. Implementation and Experiences

As of today, our implementation is integrated into a development version of TotalView and exposed through its command line interface. The corresponding MPI introspection library, has been added to Open MPI. TotalView interacts with this library.

Further implementation is an on going effort in both TotalView and Open MPI. We can successfully extract information from `MPI_Comm` handles and present it to the user. The project has also revealed the difficulty of implementing an interface, for which there is no reference implementation or existing test suite. The library is implemented "blind" by one developer. Another developer extends TotalView to support the library. While this makes different interpretations of the specification very clear, it is also time consuming. MPI vendors will need to ensure that they keep their introspection libraries up to date if their MPI implementations change.

Figure 4 shows a debugging session. It starts by loading the debug library and then performing two queries. The first

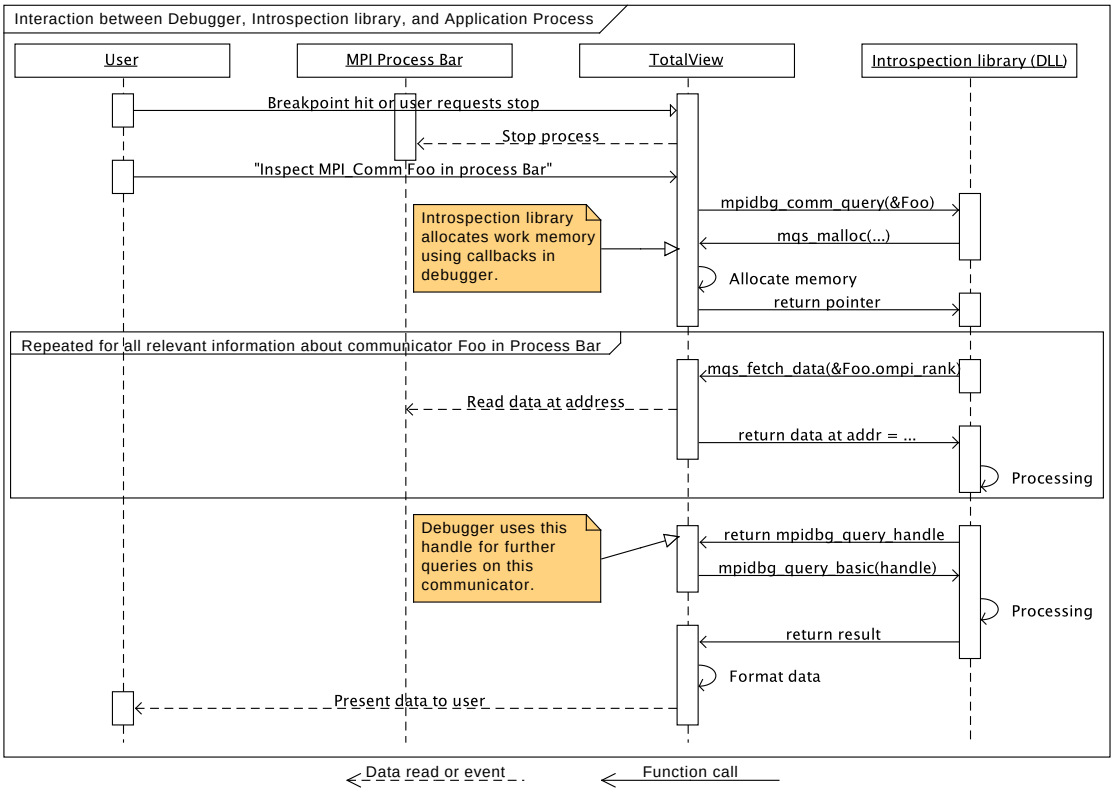

Fig. 3. Sequence diagram for querying a communicator.

```
d1.<> .mpidbg
Loaded MPI support library /g/g90/laustbn/local/lib/
openmpi/libompi_dbg_mpihandles.so :
Open MPI handle interpretation support for parallel
debuggers compiled on Sep 5 2014

Finished loading MPI introspection support.

d1.<> dfocus p2
p2.<
p2.<> .mpidbgdump
Name                                Handle
MPI_COMM_WORLD                      0x6028a0
MPI_COMM_SELF                       0x2aaaab01aa00
MPI_COMM_PARENT                     0x2aaaab01a9e0
MPI_COMM_NULL                       0x2aaaab01a3e0

p2.<> .mpidbgquery basic 0x6028a0
Querying communicator 0x6028a0 in process 0x4878780
Communicator: MPI_COMM_WORLD
Rank: 0
Size: 4
Flag                                Value
MPIDBG_COMM_INFO_PREDEFINED         True
MPIDBG_COMM_INFO_CARTESIAN          False
MPIDBG_COMM_INFO_GRAPH              False
MPIDBG_COMM_INFO_TOPO_REORDERED     False
MPIDBG_COMM_INFO_INTERCOMM          False
{...}
Query was successful
```

Fig. 4. A TotalView command line debugging session. The implementation is work in progress and can currently fetch and decode `MPI_Comm` information.

query shows Open MPI's mapping of internal communicators. The second query provides basic information on the built-in `MPI_COMM_WORLD` communicator.

IV. Conclusions and Future Work

We have successfully implemented the proposed Handle Introspection specification in TotalView and Open MPI.

The benefit for MPI developers is clear. Introspection enables high level debugging of MPI applications, without deep knowledge of the MPI library implementation. For MPI implementation vendors, it decouples the internals from the debugging, and even allows flexible debugging of closed source MPI implementations. Further work will include integration into the graphical interface of TotalView and support for additional MPI handle types.

Acknowledgments

This work (LLNL-CONF-660001) was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344, and co-funded by the Artemis PaPP Project nr. 295440 and COPCAMS project nr. 332913.

References

[1] "MPI Forum Tools Working Group," https://svn.mpi-forum.org/trac/mpi-forum-web/wiki/MPI3Tools, October 2014.

[2] Rogue Wave Software. Inc., "TotalView Graphical Debugger," 2014, http://www.roguewave.com/products/totalview.aspx, October 2014.
[3] R. L. Graham, T. S. Woodall, and J. M. Squyres, "Open MPI: A Flexible High Performance MPI," in *Parallel Processing and Applied Mathematics*. Springer, 2006, pp. 228–239.
[4] J. Cownie and W. Gropp, "A Standard Interface for Debugger Access to Message Queue Information in MPI," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Springer, 1999, pp. 51–58.
[5] J. Cownie, J. DelSignore Jr, B. R. de Supinski, and K. Warren, "DMPL: an OpenMP DLL Debugging Interface," in *OpenMP Shared Memory Parallel Programming*. Springer, 2003, pp. 137–146.
[6] A. Eichenberger, J. Mellor-Crummey, M. Schulz, N. Copty, J. DelSignore, R. Dietrich, X. Liu, E. Loh, and D. Lorenz, "OMPT and OMPD: OpenMP Tools Application Programming Interfaces for Performance Analysis and Debugging," Tech. Rep., 2013.