An Automatic MPI ABI Translation Library Builder to Enable MPI Application Binary Portability

Shinji Sumimoto
sumimoto@cc.u-tokyo.ac.jp
The University of Tokyo
Japan

Toshihiro Hanawa
hanawa@cc.u-tokyo.ac.jp
The University of Tokyo
Japan

Kengo Nakajima
nakajima@cc.u-tokyo.ac.jp
The University of Tokyo/RIKEN CCS
Japan

ABSTRACT

This paper proposes an automatic MPI ABI (Application Binary Interface) translation library builder named MPI-Adapter2. The container-based job environment is becoming wide-use in computer centers. However, when a user uses the container image in another computer center, the container with MPI binary may not work because of the difference in the ABI of MPI libraries. The MPI-Adapter2 enables to build of MPI ABI translation libraries automatically from MPI libraries. MPI-Adapter2 can build MPI ABI translation libraries not only between different MPI implementations, such as Open MPI, MPICH, and Intel MPI but also between different versions of MPI implementation. We have implemented and evaluated MPI-Adapter2 among several versions of Intel MPI, MPICH, MVAPICH, and Open MPI, and found that MPI-Adapter2 worked fine except for Open MPI ver. 4 binary on Open MPI ver. 2, because of the difference in MPI object size.

KEYWORDS

MPI Bbinary Execution Portability, MPI ABI Translation, Automatic library builder

ACM Reference Format:
Shinji Sumimoto, Toshihiro Hanawa, and Kengo Nakajima. 2023. An Automatic MPI ABI Translation Library Builder to Enable MPI Application Binary Portability. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference EuroMPI '23)*. ACM, New York, NY, USA, 5 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The next generation of computer center computing will use cloud and on-premise center systems, and users can run their jobs to multiple computer centers and cloud systems on demand without modifying the application execution environment. To realize the execution environment, container-based job execution is needed for execution portability between the cloud and the center systems. Container technology enables OS and system software portability. However, MPI libraries do not always have binary compatibility among various HPC systems. Application Binary Interface (ABI) or execution environment depends on their implementations.

To realize the ABI portability, there are MPI ABI translation libraries such as MPI-Adapter[1], wi4mpi[2], and MPItranporine[3]. However, these related works need to describe or implement all MPI functions. The MPI standard has been advancing to version 4.0. The number of MPI functions and objects is increasing, e.g., MPI 1.1=130, MPI 2.2=299, MPI 3.1=412, and MPI 4.0=467 functions.[1] Also, each implementation has extended ones, e.g., MPIX_ prefixes. So this is not easy to apply many combinations of MPI library combinations.

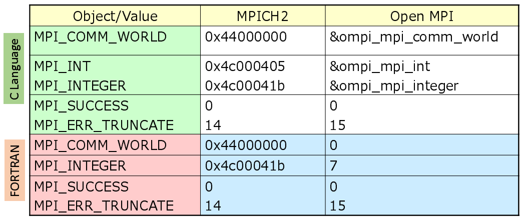
It is not easy to build them by hand. It should be built automatically. Therefore, we started to develop MPI-Adapter2.

We have developed MPI-Adapter2. It generates whole combinations of MPI adapters from MPI header files and their implementations automatically. This paper presents the MPI-Adapter2.

2 THE DESIGN OF MPI-ADAPTER2

The goal of MPI-Adapter2 is to build MPI ABI translation libraries automatically. In this sense, how to make them automatically is an issue to solve. There are two types of automatic building methods to develop. One is to build MPI ABI translation libraries, and the other is to make multiple combinations of the libraries. This section discusses the two types of methods.

2.1 Building MPI ABI Translation Libraries Automatically



	Object/Value	MPICH2	Open MPI
C Language	MPI_COMM_WORLD	0x44000000	&ompi_mpi_comm_world
	MPI_INT	0x4c000405	&ompi_mpi_int
	MPI_INTEGER	0x4c00041b	&ompi_mpi_integer
	MPI_SUCCESS	0	0
	MPI_ERR_TRUNCATE	14	15
FORTRAN	MPI_COMM_WORLD	0x44000000	0
	MPI_INTEGER	0x4c00041b	7
	MPI_SUCCESS	0	0
	MPI_ERR_TRUNCATE	14	15

Figure 1: MPI Object Value of MPICH2 and Open MPI

To automatically build an MPI ABI translation library, we need to know the MPI function prototype definitions and MPI object information (ex. MPI_Comm and MPI_Datatype) from each MPI implementation because the numbers of functions and the MPI objects implementation may differ from others.

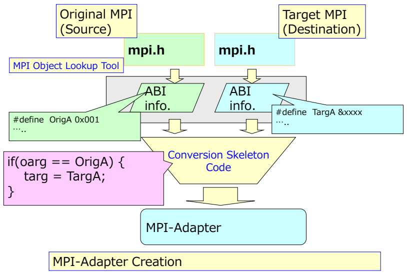
Figure 1 shows an example of conversion between Open MPI objects and MPICH2 MPI objects. Looking at the figure 1 we can see that the C and FORTRAN languages can have different implementations. In C language, MPICH2 implements as an int type, but

Conference EuroMPI '23, Sept. 11–13, 2023, Bristol, UK

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

[1] by counting begin{funcdef} in MPI forum original TeX source

Open MPI implements as a pointer to a structure. In FORTRAN, MPICH2 and Open MPI are both implemented with int type, but with different values for the same MPI Objects.



```
#define  OrigA 0x001
…..
```

```
#define  TargA &xxxx
…..
```

```
if(oarg == OrigA) {
   targ = TargA;
}
```

Figure 2: Implementation Process of MPI-Adapter

The MPI-Adapter2 also uses the MPI-Adapter method to convert the implementation differences of MPI objects. It then extracts and substitutes the values of individual MPI objects from the MPI implementation. The MPI-Adapter method worked fine among Open MPI, Intel MPI, MPICH, MVAPICH, and HP MPI libraries.[1]

Figure 2 shows the implementation process of MPI-Adapter. As an ABI conversion library function, the Conversion-Skeleton Code in the center of Fig. 2 is developed manually, and the original(source) and target(destination) ABI information is acquired programmatically from "mpi.h" and the MPI implementation and converted. However, we have to build the Conversion-Skeleton Code and to get ABI information automatically for MPI-Adapter2.

2.2 Applying Many Combination of MPI Versions and MPI implementations

2.2.1 Missmatching MPI versions between Original and Target.

Users usually develop their MPI programs with various MPI implementations and their versions, then compile them into executable binaries and execute them. In these cases, old applications used old standards at the time, and new applications used MPI functions of the latest standards.

When we use an MPI-ABI conversion library, problems may occur due to the combination of the MPI standards of the application development environment and execution environment.

If the application development environment and execution environment have the same MPI version, or if the target is a newer MPI specification, the application can run on the upper version of the MPI implementation. However, if the MPI standard of the execution environment is older than the development environment one, problems may occur.

Whether some problem occurs or not depends on the MPI functions used by the application. There are two ways to avoid them. The first way is extracting what MPI functions the application binary used and examined. The second way is detecting when an MPI application calls the functions during application runtime. However, extracting and examining the MPI functions used from the application execution binary causes a delay in the start of execution.

Therefore, we decided to use the second way. If an MPI application calls un-implemented MPI functions, MPI-AdapterII outputs an error and stops the program.

2.2.2 Missmatching Runtime Library Implementations between Original and Target.

MPI implementations currently exist in Open MPI-based and MPICH-based implementations.

Regarding the Open MPI implementation, the basic MPI Object definition is the same, but it depends on Open MPI versions. If A is different in the definition of version A.B.C., Open MPI may not keep backward compatibility.[5]

Although the basic MPI Object definition is the same, there are problems at runtime due to changes in the configuration of the dynamic link library which are different names and differences in the structure sizes in the MPI-Object in C language. There is a problem between Open MPI ver.2 and Open MPIver.3 or later because some MPI-Object structure sizes are different. In this case, there is a problem that an MPI program will crash.

In the case of the MPICH system, there is a WG to keep ABI among MPICH and derived implementations, and the WG says that they will mutually keep ABI compatibility. [4]

2.3 Design Overview

To develop MPI-Adapter2 using MPI library information, it is necessary to know the list of MPI functions and the prototype definition in C language.

For this reason, for C language, the implementation MPI function prototypes and MPI-Objects of each MPI library are automatically generated from mpi.h in principle. However, some Open MPI internal implementations are unknown in mpi.h and are generated individually. (e.g. size of MPI object structure of Open MPI)

As for FORTRAN, there is mpif.h, but it does not contain the prototype definition like in C language, so the API specification is created by extracting it from the source code of the MPI library implementation.

The procedures for realizing mpi-adapter2 are shown below.

(1) Creates a list of MPI libraries generated by MPI-Adapter2. This list describes the MPI header information, the location of the mpicc, and the nickname.
(2) Extracts MPI(PMPI, MPIX) API prototype and MPI object definitions for C language mpi.h (For FORTRAN language, from MPI library implementation)
(3) Automatically generates conversion sources (Conversion Skeleton Code) from conversion-source MPI to conversion-destination MPI for each argument of each prototype definition. At the same time, checks whether each argument is an MPI object or not, and finds the definition from source and destination MPI implementation.
(4) In the Conversion Skeleton code, there are exceptions depending on the function, so deal with them separately. (MPI_Alltoallw, MPI_Waitall, MPI_Startall, etc.)
(5) Functions in the ABI conversion list pre-define for each MPI object, and expand using symbols with the prefixes of the original(source) and target(destination) MPI library implementations added.
(6) Prepares a function array (MPI global function array) by merging and ORing MPI function names from all MPI implementations to be converted in the MPI library lists.
(7) The MPI global functions array contains function pointers to the destination MPI functions in the Conversion Skeleton

Code generate code that calls the functions in the MPI global functions array. At the MPI-Adapter2 program initialization, the MPI-Adapter2 extracts the provided MPI functions from the destination MPI library symbol information. If an MPI function exists, MPI-Adapter2 stores the MPI function pointers to the MPI global function array. If it is not registered, the MPI-Adapter2 stores the function pointer which outputs an error and exits.

(8) Outputs Makefiles to build MPI-Adapters with the combinations of MPI library implementations.

(9) Generates MPI library implementation definitions automatically to make MPI-Adapters by the make command.

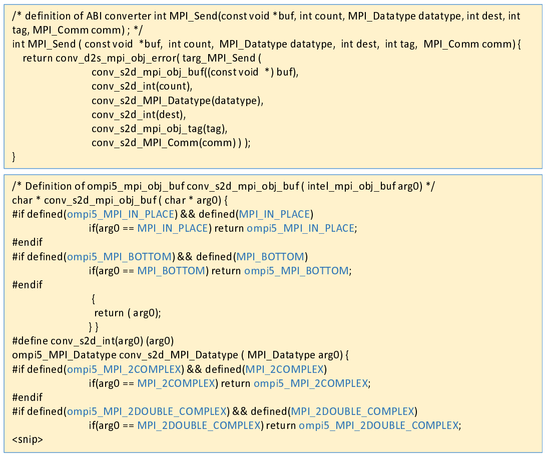
3 MPI ADAPTER2 IMPLEMENTATION

To automatically generate the MPI-Adapter2 code for each MPI library, we implemented MPI-Adapter2 using the scripting language Perl for text processing. Although the implementation is different for both the C and the FORTRAN languages, we implemented using the same processing method.

MPI-Adapter2 automatically generates conversion sources (Conversion Skeleton Code) of conversion-source MPI and conversion-destination MPI for each argument of each prototype definition. An outline of this conversion is as follows.

- After parsing the arguments of the MPI prototype definition and outputting the ABI conversion code to call the conversion-destination MPI function
- ABI conversion function arguments from conversion-source MPI to conversion-destination MPI are implemented when the arguments are not pointers, and exceptions such as arrays are handled by individual patterns.
- ABI conversions of return values and pointers from conversion-destination to conversion-source are processed after calling the destination MPI function.
- Return the return value after ABI conversion

Fig. 3 shows the sample output of MPI ABI Conversion Skeleton Codes.

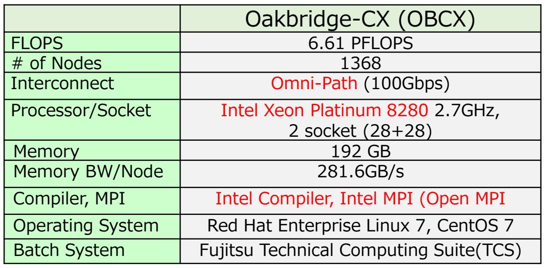


```
/* definition of ABI converter int MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest, int
tag, MPI_Comm comm) ; */
int MPI_Send ( const void  *buf,  int count,  MPI_Datatype datatype,  int dest,  int tag,  MPI_Comm comm) {
   return conv_d2s_mpi_obj_error( targ_MPI_Send (
                         conv_s2d_mpi_obj_buf((const void  * ) buf),
                         conv_s2d_int(count),
                         conv_s2d_MPI_Datatype(datatype),
                         conv_s2d_int(dest),
                         conv_s2d_mpi_obj_tag(tag),
                         conv_s2d_MPI_Comm(comm) ) );
}
```

```
/* Definition of ompi5_mpi_obj_buf conv_s2d_mpi_obj_buf ( intel_mpi_obj_buf arg0) */
char * conv_s2d_mpi_obj_buf ( char * arg0) {
#if defined(ompi5_MPI_IN_PLACE) && defined(MPI_IN_PLACE)
                        if(arg0 == MPI_IN_PLACE) return ompi5_MPI_IN_PLACE;
#endif
#if defined(ompi5_MPI_BOTTOM) && defined(MPI_BOTTOM)
                        if(arg0 == MPI_BOTTOM) return ompi5_MPI_BOTTOM;
#endif
                         {
                          return ( arg0);
                        } }
#define conv_s2d_int(arg0) (arg0)
ompi5_MPI_Datatype conv_s2d_MPI_Datatype ( MPI_Datatype arg0) {
#if defined(ompi5_MPI_2COMPLEX) && defined(MPI_2COMPLEX)
                        if(arg0 == MPI_2COMPLEX) return ompi5_MPI_2COMPLEX;
#endif
#if defined(ompi5_MPI_2DOUBLE_COMPLEX) && defined(MPI_2DOUBLE_COMPLEX)
                        if(arg0 == MPI_2DOUBLE_COMPLEX) return ompi5_MPI_2DOUBLE_COMPLEX;
<snip>
```

Figure 3: Sample Output of MPI ABI Conversion Skeleton Codes

4 EVALUATION OF MPI-ADAPTER2

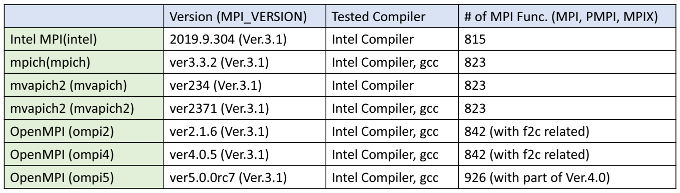


	Oakbridge-CX (OBCX)
FLOPS	6.61 PFLOPS
# of Nodes	1368
Interconnect	Omni-Path (100Gbps)
Processor/Socket	Intel Xeon Platinum 8280 2.7GHz, 2 socket (28+28)
Memory	192 GB
Memory BW/Node	281.6GB/s
Compiler, MPI	Intel Compiler, Intel MPI (Open MPI
Operating System	Red Hat Enterprise Linux 7, CentOS 7
Batch System	Fujitsu Technical Computing Suite(TCS)

Figure 4: Evaluation Enviromnent

As the evaluation of MPI-Adapter2, we evaluated MPI ABI mutual portability using the PingPong benchmark and NAS Parallel Benchmarks. We also evaluated MPI ABI portability in a container environment using Singularity. The system used as the evaluation environment was Oakbridge-CX (OBCX) of the Information Technology Center of the University of Tokyo (Fig. 4).

Fig. 5 shows the MPI library implementations used for evaluation. The rightmost column in Table 3 shows the number of functions developed by MPI-Adapter2. The number of functions defined in MPI Ver.3.1 is 412, but the number of functions expanded has increased because some MPI functions, such as the PMPI prefix MPIX, were also included. In this evaluation, we could not prepare the TCS-compatible OmniPath MPI library implementation due to an install problem, so we evaluated it on a single node using a shared memory device.



	Version (MPI_VERSION)	Tested Compiler	# of MPI Func. (MPI, PMPI, MPIX)
Intel MPI(intel)	2019.9.304 (Ver.3.1)	Intel Compiler	815
mpich(mpich)	ver3.3.2 (Ver.3.1)	Intel Compiler, gcc	823
mvapich2 (mvapich)	ver234 (Ver.3.1)	Intel Compiler	823
mvapich2 (mvapich2)	ver2371 (Ver.3.1)	Intel Compiler, gcc	823
OpenMPI (ompi2)	ver2.1.6 (Ver.3.1)	Intel Compiler, gcc	842 (with f2c related)
OpenMPI (ompi4)	ver4.0.5 (Ver.3.1)	Intel Compiler, gcc	842 (with f2c related)
OpenMPI (ompi5)	ver5.0.0rc7 (Ver.3.1)	Intel Compiler, gcc	926 (with part of Ver.4.0)

Figure 5: Evaluation List of Target MPI Implementations

4.1 MPI PingPong Benchmark

In this section, we use the MPI PingPong benchmark to measure the overhead of MPI ABI conversion. Due to space limitations, we show the results for Open MPI ver.5.0 and Intel MPI.

Figure 6 shows the results when using various MPI libraries as source MPI running on Open MPI ver5.0 as destination MPI. From Fig. 6, we can see that the performance trend is the same as Open MPI ver5.0, and the overhead of MPI ABI conversion is a maximum of 30ns and negligible.

Figure 7 shows the results when using various MPI libraries running on Intel MPI. From Fig. 7, we can see that the performance trend is the same as Intel MPI, and the overhead of MPI ABI conversion is a maximum of 20ns and negligible.

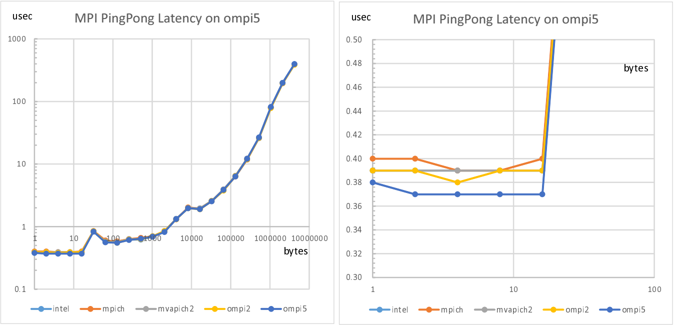


Figure 6: MPI PingPong Benchmark on Open MPI ver5.0

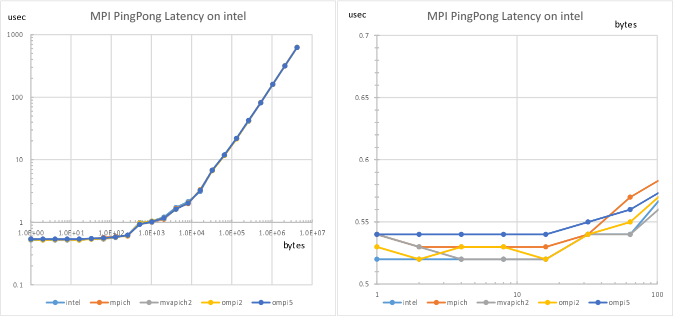


Figure 7: MPI PingPong Benchmark on Intel MPI

4.2 NAS Parallel Benchmark

n this section, we show the results of mutual execution using the NAS Parallel Benchmark to evaluate the ABI execution portability of MPI-Adapter2.

The NAS Parallel Benchmarks is a code created as a NASA system benchmark and consists of eight types of programs: BT, CG, EP, FT, IS, LU, MG, and SP. Only IS is written in C language and the other seven programs are FORTRAN.

We evaluated all NAS Parallel Benchmark (Class A) programs with the number of processes from 1 to 32 (BT, SP up to 25) and all combinations of MPI library implementations.

As a result of the evaluation, only the IS benchmark ompi4 binary failed with a segmentation fault in the ompi2 environment (5 programs of IS (2, 4, 8, 16, 32 processes) out of all 46 benchmarks).

In the other cases, we confirmed that the calculation results were correct for all benchmarks and all combination sets of MPI libraries. The reason is the difference in structure sizes that make up MPI Object in C language. In addition, ompi5 does not have a problem because ompi5 changed the structure size to the same as ompi2.

Among these execution results, as evaluation examples, the conversion destinations are shown in Figures 8, 9, 10, and 11, respectively. These results also include the results of native execution and are the results of up to 32 processes on one node of OBCX.

In the evaluation of ABI conversion to Open MPI ver.5 in Fig. 8 and Fig. 9, although the compiled MPI execution environment is different, the execution results show almost the same tendency, and there is no particularly slow result compared to native execution.

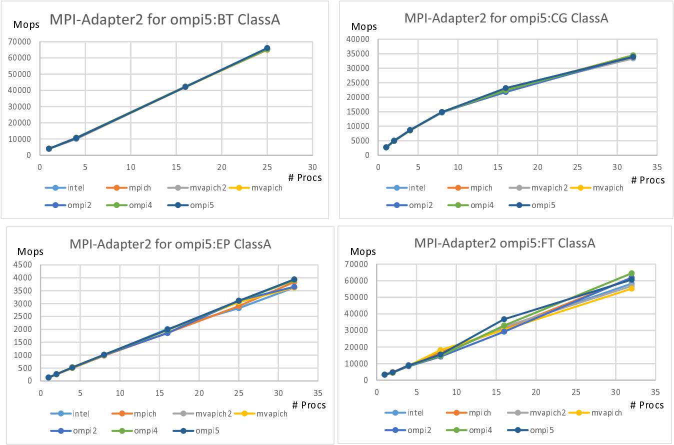


Figure 8: NAS Parallel Benchmarks on Open MPI v5.x(1)

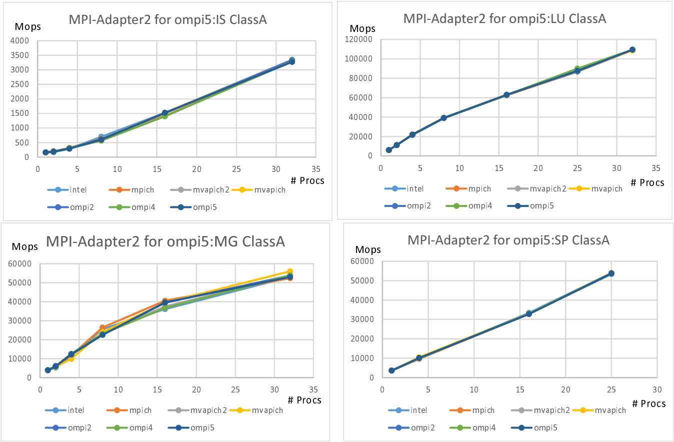


Figure 9: NAS Parallel Benchmarks on Open MPI v5.x(2)

In the evaluation of ABI conversion to Intel MPI in Fig. 10 and Fig. 11, the execution results show almost the same tendency even though the compiled MPI execution environment is different, and no particularly slow result was seen compared to native execution.

4.3 MPI ABI Portability Evaluation on Container

This section evaluates ABI portability in the Container environment. I used Singularity as a Container and created a Container using Ubuntu (Ubuntu 22.04.1 LTS). This time, we use Singularity as a Container with Open MPI 4.0.5 and PMB (Pallas MPI Benchmark) on Ubuntu 22.04.1 LTS, and ABI portability is evaluated using MPI-Adapter2.

The modules libelf-dev, libpciaccess0, libxml2-dev, gfortran, and libquadmath0 were required when running Open MPI executable binaries on MPICH (gcc+gfortran) on the host. For this reason, We added these modules to the Singularity image with apt-get and ran it. (Container image: ubuntu.sif)

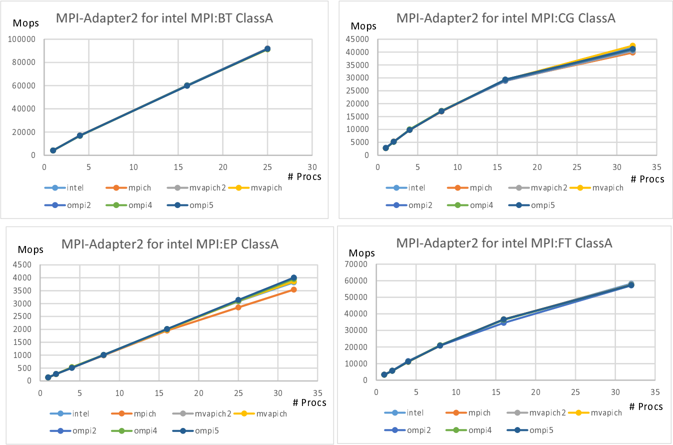


Figure 10: NAS Parallel Benchmarks on Intel MPI(1)

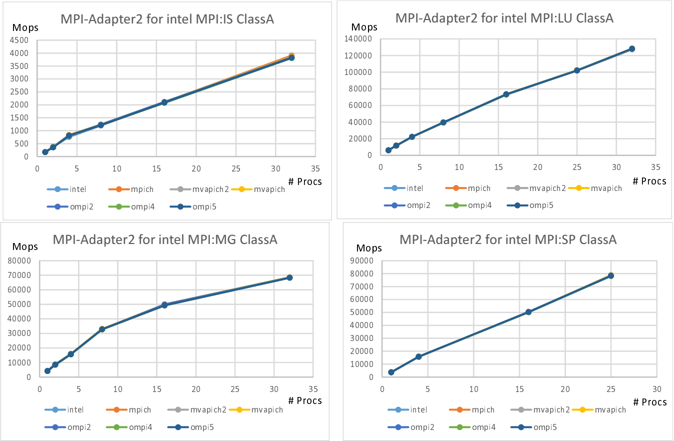


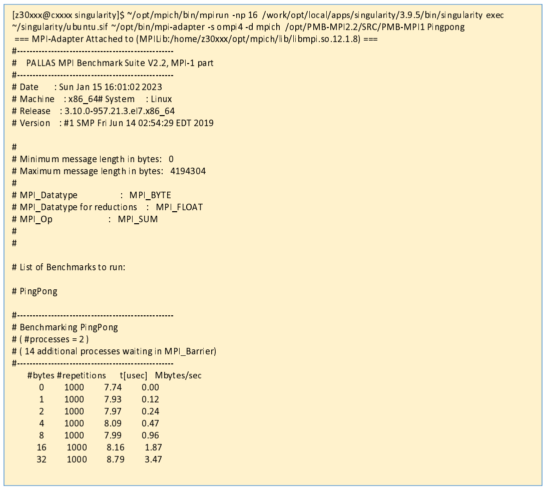
Figure 11: NAS Parallel Benchmarks on Intel MPI(2)

Figure 12 shows the execution command line and running result. At the execution, we can run the PMB benchmark execution binary by the Open MPI ver. 4 mpicc on the MPICH host execution environment using the command line " /opt/mpich/bin/mpirun -np 16 /work/opt/local/apps/singularity/3.9.5/bin/singularity exec /singularity/ubuntu.sif /opt/bin/mpi-adapter -s ompi4 -d mpich /opt/PMB-MPI2.2/SRC/PMB-MPI1 Pingpong".

We can see that the execution binary of Open MPI on Container (ubuntu.sif) works well with MPICH runtime on the host.

5 SUMMARY

This paper describes MPI-Adapter2 which keeps ABI portability between MPI libraries. MPI-Adapter2 automatically generates an ABI conversion library from the mpi.h file for ABI compatibility between these MPI libraries. As a result of evaluation with Intel MPI, MPICH, MVAPICH (2 types), and Open MPI (3 types), we showed that all kinds of NAS Parallel Benchmark can run well while ensuring mutual ABI compatibility. Also, we found that MPI-Adapter2 worked fine using Singularity container images and shows that container images with MPI binaries can keep binary portability among computer centers with different MPI libraries.


```
[z30xxx@cxxxx singularity]$ ~/opt/mpich/bin/mpirun -np 16  /work/opt/local/apps/singularity/3.9.5/bin/singularity exec
~/singularity/ubuntu.sif ~/opt/bin/mpi-adapter -s ompi4 -d mpich  /opt/PMB-MPI2.2/SRC/PMB-MPI1 Pingpong
 === MPI-Adapter Attached to (MPILib:/home/z30xxx/opt/mpich/lib/libmpi.so.12.1.8) ===
#---------------------------------------------------
#    PALLAS MPI Benchmark Suite V2.2, MPI-1 part
#---------------------------------------------------
# Date       : Sun Jan 15 16:01:02 2023
# Machine    : x86_64# System     : Linux
# Release    : 3.10.0-957.21.3.el7.x86_64
# Version    : #1 SMP Fri Jun 14 02:54:29 EDT 2019

#
# Minimum message length in bytes:   0
# Maximum message length in bytes:   4194304
#
# MPI_Datatype                   :   MPI_BYTE
# MPI_Datatype for reductions    :   MPI_FLOAT
# MPI_Op                         :   MPI_SUM
#
#

# List of Benchmarks to run:

# PingPong

#---------------------------------------------------
# Benchmarking PingPong
# ( #processes = 2 )
# ( 14 additional processes waiting in MPI_Barrier)
#---------------------------------------------------
       #bytes #repetitions      t[usec]   Mbytes/sec
            0         1000         7.74         0.00
            1         1000         7.93         0.12
            2         1000         7.97         0.24
            4         1000         8.09         0.47
            8         1000         7.99         0.96
           16         1000         8.16         1.87
           32         1000         8.79         3.47
```

Figure 12: MPI-Adapter2 Execution Result on Singularity

The method used in this paper was applied to MPI this time, but we can apply it to other libraries.

For future works, we will proceed with practical evaluations with more application binaries. We will also make use of MPI-Adapter2 not only to ensure ABI compatibility but also to expand the replacement functions of unsupported functions in the conversion destination in a plug-in format as an extension of user convenience. (Example: MPI_Count support), and replacing the specified MPI function with a user-written function (e.g., MPI collective communication algorithm).

REFERENCES

[1] Shinji Sumimoto, Kohta Nakashima, Akira Naruse, Kouichi Kumon, Takashi Yasui, Yoshikazu Kamoshida, Hiroya Matsuba, Atsushi Hori, and Yutaka Ishikawa. The design of seamless mpi computing environment for commodity-based clusters. In Matti Ropo, Jan Westerholm, and Jack Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 9–19, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[2] Edgar A. León, Marc Joos, Nathan Hanford, Adrien Cotte, Tony Delforge, François Diakhaté, Vincent Ducrot, Ian Karlin, and Marc Pérache. On-the-fly, robust translation of mpi libraries. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 504–515, 2021.

[3] MPItrampoline: https://github.com/eschnett/MPItrampoline.

[4] MPICH ABI: https://www.mpich.org/abi/.

[5] Open MPI SC21 BoF: https://www-lb.open-mpi.org/papers/sc-2021/Open-MPI-SC21-BOF.pdf.