Working with Multiple MPIs

Overcoming ABI Incompatibility

Edgar León
Laurent Nguyen
Nathan Hanford

Marc Joos
Eric Green
Adrien Cotte
Clément Fontenaille

21 May 2023

IMAGINE TOMORROW
ISC High Performance
MAY 21 – 25, 2023 | HAMBURG, GERMANY

LLNL-PRES-848786.
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344.
Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

Wi4MPI
Wrapper interface for MPI

14:00	14:35	**Introduction and setup**
14:35	16:00	**Changing MPIs dynamically with Wi4MPI** • **Spack setup** • **Preload and interface mode**
16:00	16:30	***Café***
16:30	18:00	**Applying Wi4MPI to HPC** • **MPI for Python** • **Mini-application exploration** • **Podman**

https://github.com/LLNL/productivity-frameworks/tree/main/tutorials/isc23

What if MPI applications could use any MPI library at runtime without recompiling?

- Working around limitations of an MPI library
  - Help diagnose the source of a problem
  - Choose the best MPI
- Enabling fast/portable containers
  - Containers provide flexibility and portability
  - Loss of portability to match the host MPI library
- Adding flexibility to high-level languages
  - High-level languages can depend on a specific MPI library
- Running on bleeding-edge/early-access systems
  - State-of-the-art systems may come with a single, vendor-optimized library

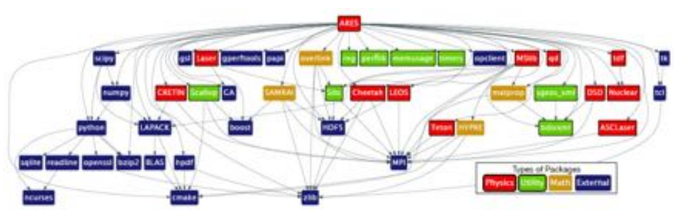
An MPI library may use its own Application Binary Interface!

- MPI has a single API 😃
- MPI may have several ABIs 😫
  - OpenMPI, MPICH, MPC

```
typedef int MPI_Comm;
#define MPI_COMM_WORLD ((MPI_Comm) 0x44000000)
```

```
typedef struct ompi_communicator_t *MPI_Comm;
#define MPI_COMM_WORLD ((MPI_Comm) &ompi_mpi_comm_world)
```

- Need to recompile to use a different MPI library
  - May or may not be feasible



Gamblin et al. The Spack package manager: bringing order to HPC software chaos. SC'15.

Wi4MPI: A general approach to ABI translation

- *T*: $f_\beta \rightarrow f_\rho$
  1. Translate input arguments from the β ABI to the ρ ABI
  2. Call f_ρ
  3. Translate output arguments and return value from the ρ ABI to the β ABI
- Automatically generate translation functions from two JSON files
  - The **functions**' signatures
  - The **mappers**' descriptor used to translate arguments

Function	*name*	Symbol's name
	args	Hash describing each argument
	return	Return mapping
Func. Argument	*name*	Argument's name
	mapper	Mapping descriptor to translate arg.
	in	Should argument be converted before *f*?
	out	Should argument be converted after *f*?
	size	Name of size argument for non-scalars
Mapper	*type*	Argument's type
	b2r	Func. to translate argument from β to ρ
	r2b	Func. to translate argument from ρ to β
	alloc	Should argument be allocated locally?

int MPI_Comm_rank(MPI_Comm comm, int* rank)

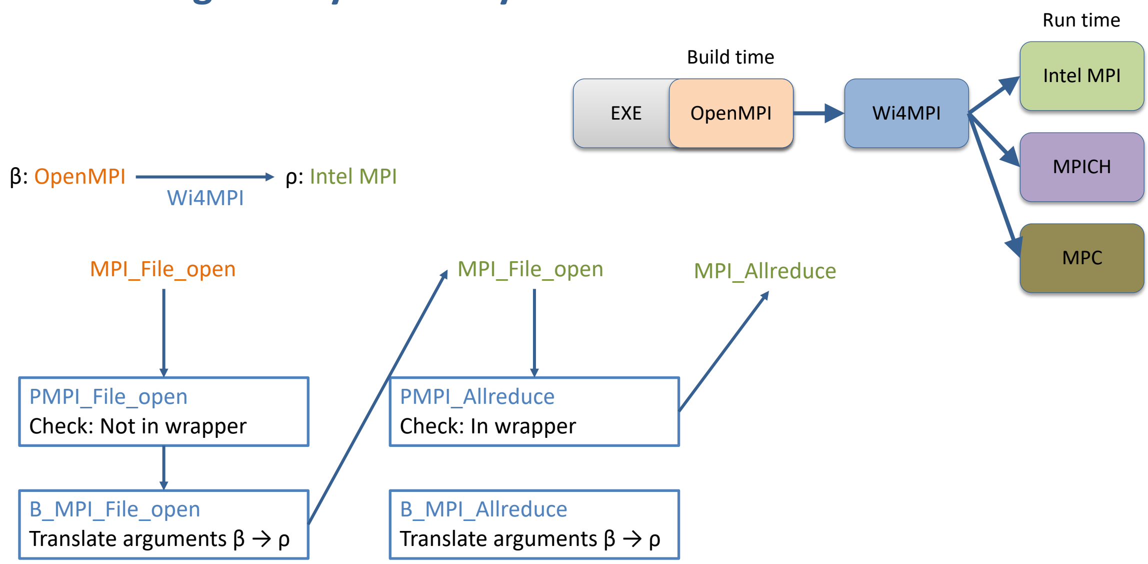
```
name: MPI_Comm_rank,

args: [
  {  name: comm,
   mapper: comm_converter,
       in: 1,
      out: 0
  },
  {  name: rank,
   mapper: int_ptr_mapper,
       in: 0,
      out: 1
  }
],

return: {
    name: rc
  mapper: error_converter,
      in: 0,
     out: 1
}...
```

Function JSON file

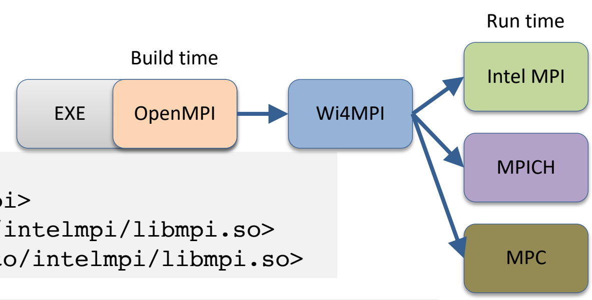
```
comm_converter: {
  type: MPI_Comm,
  b2r: comm_conv_b2r,
  r2b: comm_conv_r2b,
  alloc: 0
}...
```

Mapper JSON file

Translating MPI dynamically



Wi4MPI is easy to use



1

```
# Specify Intel MPI as the target
export INTELMPI_ROOT=<path/to/intelmpi>
export WI4MPI_RUN_MPI_C_LIB=<path/to/intelmpi/libmpi.so>
export WI4MPI_RUN_MPIIO_C_LIB=<path/to/intelmpi/libmpi.so>

# Specify the Wi4MPI library to use
export LD_PRELOAD=<path/to/wi4mpi>/libwi4mpi_openmpi_intelmpi.so
```

1

```
# Load the appropriate Wi4MPI module
module load wi4mpi/3.4.0_openmpi_to_intelmpi
```

2

```
# Launch binary built with OpenMPI
srun –N20 –n80 <app_binary>
```

Use cases demonstrating Wi4MPI

- Choosing the best MPI for my code
  - Arm architecture
  - x86 architectures
  - x86 + GPUs hybrid architectures
- Enabling applications on new hardware
  - New GPU systems
  - Bleeding-edge and early-access systems
- Running fast and portable containers
  - Use any MPI with host-tuned performance

On-the-Fly, Robust Translation of MPI Libraries

Edgar A. León*, Marc Joos†, Nathan Hanford*, Adrien Cotte‡, Tony Delforge‡,
François Diakhaté†, Vincent Ducrot‡, Ian Karlin*, and Marc Pérache†
*Lawrence Livermore National Laboratory, Livermore, CA, USA
{leon, nhanford, karlin1}@llnl.gov
†CEA,DAM,DIF F-91297 Arpajon, France
{marc.joos, francois.diakhate, marc.perache}@cea.fr
‡AS+ Groupe Eolen, 37-39 rue Boissière, Paris, France
{adrien.cotte, tony.delforge, vincent.ducrot}@eolen.com

***Abstract*—Most parallel scientific applications rely on third-party libraries, some of which may have multiple implementations including open-source and vendor-proprietary. While sharing an application programming interface (API), many of these implementations do not have a shared application binary interface (ABI) and require recompiling applications to change the library implementation used. For many applications, recompiling is a long and complex process and sometimes not even an option when the application is shipped binary only. ABI incompatibility strikes at the heart of portability, productivity, and performance by (1) impeding application execution across different systems; (2) adding developer hours rebuilding an application; and (3) not taking advantage of host-optimized libraries.**

In this paper, we present a methodology and framework to solve ABI incompatibility across MPI libraries, which follow a well-defined API. The proposed framework called Wi4MPI translates the ABI dynamically from the MPI library used to build the application to a different MPI library available at run time. We show Wi4MPI works robustly on a wide spectrum of architectures, networks, and MPI libraries. Furthermore, we demonstrate its usefulness on several use cases highlighting significant portability, performance, and productivity benefits.

***Index Terms*—Wi4MPI, ABI compatibility, ABI dynamic translation, portable MPI, MPI libraries.**

I. Introduction

Application developers today must balance performance, portability, and productivity because (1) high performance computing systems are becoming more diverse and (2) increasingly large code bases are needed to perform complex science calculations. Portability is not limited to application code and extends to third-party libraries and middleware that applications build upon. This middleware includes the MPI libraries HPC applications use to communicate within and across nodes. Most systems are delivered with a vendor-optimized library that provides the best performance and sometimes is the only working MPI library on day one.

This can lead to application portability challenges because, while MPI has a single Application Programming Interface (API) that all implementations use, each MPI implementation uses its own Application Binary Interface (ABI). Different ABIs create a compatibility challenge for portability [1]. For example, once an application is built and linked to a compiled library with one ABI, it is challenging, if not impossible, to *run* that application correctly with a different library that uses a *different* ABI.

ABI incompatibility requires developers to recompile their entire application to use a different MPI. These forced recompiles are costly in terms of productivity. Large applications can contain tens to hundreds of dependencies [2]–[4] and require long build times. For example, the ARES multi-physics code at Lawrence Livermore National Laboratory (LLNL) uses 47 packages with complex dependencies between them [5]. The high cost of rebuilding can discourage developers from trying alternative MPIs that may perform better or contain features the implementation they currently use does not.

In this paper, we present Wi4MPI, a solution to ABI incompatibility in the context of MPI. Wi4MPI has been in production at the French Alternative Energies and Atomic Energy Commission (CEA) since 2016 and is available as open source [6]. It allows developers to swap MPI libraries without recompiling their applications. The cost of using Wi4MPI to leverage a new MPI vs. recompiling is small in most cases, but there are instances where it adds significant overhead, particularly for small messages. However, Wi4MPI unlocks the following productivity and performance gains:

- Switch MPI libraries rapidly to work around bugs and compare performance and scalability of two libraries.
- Use a different MPI library from a container at run time, e.g., a host-optimized MPI, to access better performance without sacrificing flexibility.
- Quickly port to bleeding-edge systems and use a vendor-tuned MPI without having to compile for that MPI.
- Access multiple MPIs and better performance from high-productivity languages such as Python, which is strongly linked to a specific MPI library.

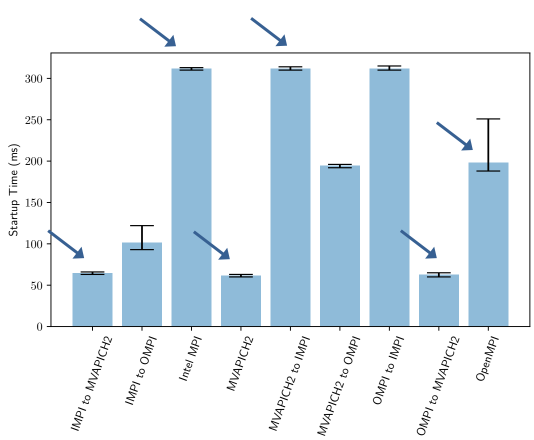
In addition to the preceding performance and productivity advantages that this paper demonstrates, the paper presents our approach to solving the ABI incompatibility problem. We take advantage of MPI having a well-defined API with known parameters to allow us to create translators that can map between the ABIs of two different implementations.

The rest of the paper is organized as follows. Section II discusses in detail selected use cases that Wi4MPI helps with. Then, Section III describes first a generic approach to ABI translation before detailing specifically how Wi4MPI does ABI translation for MPI libraries. Section IV describes

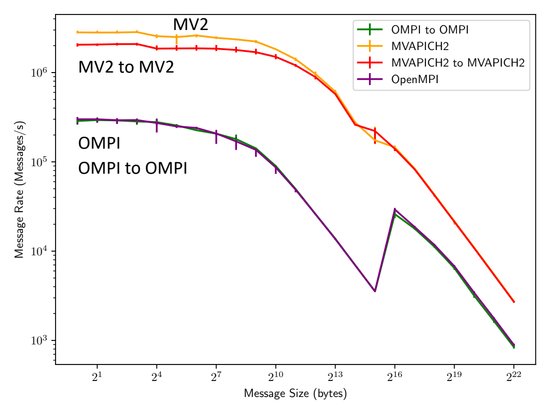
IEEE CLUSTER 2021
Portland, OR, USA • 7–10 September

Wi4MPI allows me to choose the best MPI at runtime

Performance of translated and native MPI operations using OMB on x86 nodes



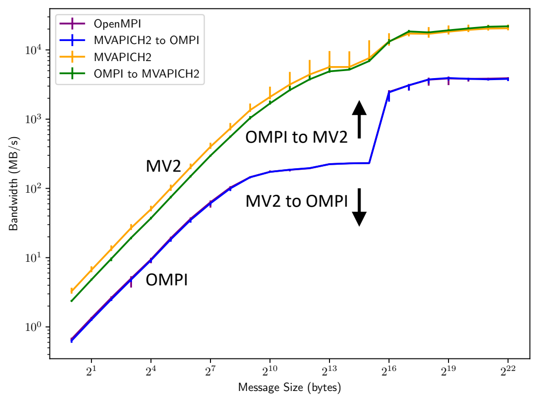
Better Startup time



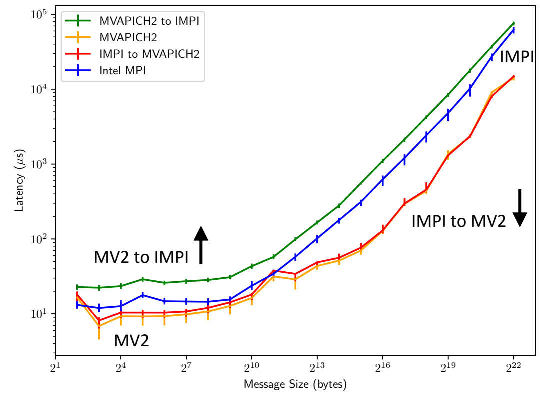
Better Messaging rate across 2 nodes

The performance gains of a better MPI often outweigh small-message overhead

Performance of translated and native MPI operations using OMB on x86 nodes



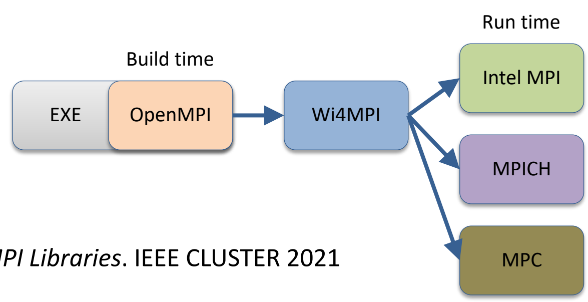
Better Bidirectional bandwidth across 2 nodes



Better Allreduce time of 144 tasks across 4 nodes

Wi4MPI solves ABI incompatibility in the context of MPI

- Wi4MPI provides an effective framework for ABI translation
- Efficiently translating MPI has many benefits
  - Choose the best MPI for your code
  - Portable and fast Containers
- Wi4MPI has limitations
  - High overhead in some cases
  - Partial support for MPI_THREAD_MULTIPLE
- Wi4MPI is open source
  - León et al., *On-the-fly, Robust Translation of MPI Libraries*. IEEE CLUSTER 2021
  - https://github.com/cea-hpc/wi4mpi



MPI Forum: Standardizing the ABI layer

- MPI Forum *likely* to define a C ABI in the future
  - Two ABIs cover over 90% of HPC platforms
- Plan is to have a single-feature ABI-only release for MPI 4.2
  - SC 2024
- MPICH has a prototype and would support the standard ABI
  - https://github.com/jeffhammond/mukautuva
- More information at the MPI ABI Working Group
  - https://github.com/mpiwg-abi

Credit: Jeff Hammond on behalf of the MPI ABI Working Group

Hands-on: AWS Parallel Cluster

- `ssh` client needed
- 35 accounts
  - `user1`, …, `user35`
- `ssh user<k>@`
- IP address:
- Username: `user<k>`
- Password:

Wrapper interface for MPI

https://github.com/LLNL/productivity-frameworks/tree/main/tutorials/isc23