Overcoming MPI ABI incompatibility

Using the Wi4MPI library to work with multiple MPIs

Marc Joos — CEA – marc.joos@cea.fr

Wrapper interface for MPI

Outline

1. Introduction

Why do we need MPI library portability?

- **Working around limitations of an MPI library**
  - help diagnose the source of a problem
  - choose the best MPI implementation
- **Enabling fast & portable containers**
  - containers provide flexibility and portability…
  - … but loss of portability to match the host MPI library
- **Adding flexibility to high-level language**
  - 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
  - sometimes also the case with cloud providers

2. Understanding ABI compatibility in MPI

An MPI library may use its own Application Binary Interface!

👍 MPI has a single API

An MPI library may use its own Application Binary Interface!

👍 MPI has a single API

👎 MPI has several ABIs

Open MPI, MPICH, MPC

An MPI library may use its own Application Binary Interface!

👍 MPI has a single API

👎 MPI has several ABIs
Open MPI, MPICH, MPC

- as a result, MPI libraries are (generally) ABI-incompatible
- this is true even for the simplest element of an MPI library:
  - MPICH:

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

  - Open MPI:

```
typedef struct ompi_communicator_t *MPI_Comm;
OMPI_DECLSPEC extern struct ompi_predefined_communicator_t ompi_mpi_comm_world;
#define MPI_COMM_WORLD OMPI_PREDEFINED_GLOBAL( MPI_Comm, ompi_mpi_comm_world)
```

An MPI library may use its own Application Binary Interface!

MPI has a single API

MPI has several ABIs
Open MPI, MPICH, MPC

- as a result, MPI libraries are (generally) ABI-incompatible
- this is true even for the simplest element of an MPI library:
  - MPICH:

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

  - Open MPI:

```
typedef struct ompi_communicator_t *MPI_Comm;
OMPI_DECLSPEC extern struct ompi_predefined_communicator_t ompi_mpi_comm_world;
#define MPI_COMM_WORLD OMPI_PREDEFINED_GLOBAL( MPI_Comm, ompi_mpi_comm_world)
```

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

3. Dynamically translating MPI libraries

Wi4MPI: switching between MPI libraries

Using a general approach to ABI translation

$$T : f_{\text{from}} \rightarrow f_{\text{to}}$$

1. input arguments translation from origin to destination ABI
2. f_{to} call
3. output arguments translation & return value from destination to origin ABI

- Prevent MPI calls triggered by ROMIO from being *re-translated* (that would result in a crash)
  - ASM code selector
- functions to pass the appropriate arguments to the underlying MPI library are generated

How to use Wi4MPI

- **There are two modes available to use Wi4MPI:**

Preload mode

- translate between MPI implementations at runtime

Interface mode

- “stub” MPI implementation, using a defined MPI implementation at runtime

- Installation:
  - CMake based installation
  - available through Spack package manager

How to use Wi4MPI

- **In practice:**
  - directly as a wrapper:

```
srun -n <nproc> wi4mpi -f mpich -t openmpi <app-binary>
```

  - transparently, using environment variables:

```
export OPENMPI_ROOT=<path to openmpi>
export WI4MPI_FROM=MPICH
export WI4MPI_TO=OMPI
export WI4MPI_RUN_MPI_C_LIB=${OPENMPI_ROOT}/lib/libmpi.so
export WI4MPI_RUN_MPI_F_LIB=${OPENMPI_ROOT}/lib/libmpi_mpifh.so
export WI4MPI_RUN_MPIIO_C_LIB=${WI4MPI_RUN_MPI_C_LIB}
export WI4MPI_RUN_MPIIO_F_LIB=${WI4MPI_RUN_MPI_F_LIB}
export LD_PRELOAD=${WI4MPI_ROOT}/libexec/wi4mpi/libwi4mpi_${WI4MPI_FROM}_${WI4MPI_TO}.so:\
                    ${WI4MPI_RUN_MPI_C_LIB}

srun -n <nproc> <app-binary>
```

How to use Wi4MPI

- **In practice:**
  - directly as a wrapper:

```
srun -n <nproc> wi4mpi -f mpich -t openmpi <app-binary>
```

  - transparently, using environment variables:

```
<exports>

srun -n <nproc> <app-binary>
```

The `WI4MPI_*` variables are listed in the documentation.

How to use Wi4MPI

- **In practice:**
  - directly as a wrapper:

```
srun -n <nproc> wi4mpi -f mpich -t openmpi <app-binary>
```

  - transparently, using environment variables:

```
<exports>

srun -n <nproc> <app-binary>
```

The `WI4MPI_*` variables are listed in the documentation.

- If the translation works, you should have this kind of output:

```
You are using Wi4MPI-3.7.0 with the mode preload From MPICH To OMPI
# OSU MPI Hello World Test v7.0
This is a test with 4 processes
```

How to use Wi4MPI & (more) advanced usage

- RTFM!
  - `https://wi4mpi.readthedocs.io`
  - 7 tutorials available (as of February 2024):
    - How to install Wi4MPI
    - Translating MPI dynamically using Preload mode
    - Translating MPI dynamically using Interface mode
    - Applying Wi4MPI to distributed Python
    - Running GROMACS with Wi4MPI
    - Applying Wi4MPI to RedHat container runtime: Podman
    - Applying Wi4MPI to a Gromacs Podman container

Wi4MPI: an open source project and on-going collaboration

- Wi4MPI started in 2016 at CEA (France) and still in active development
- Wi4MPI is open source
  - `https://github.com/cea-hpc/wi4mpi`
  - dual license CeCILL-B & BSD-3
- developments are validated using a CI including well-established benchmarks (eg. OMB, IOR, AMG, GROMACS)
- Wi4MPI is an on-going collaboration between CEA and LLNL (USA)
  - started in 2020
  - E. A. Leon, M. Joos, N. Hanford, A. Cotte, T. Delforge, F. Diakhaté, V. Ducrot, I. Karlin, M. Pérache, "On-the-Fly, Robust Translation of MPI Libraries", IEEE International Conference on Cluster Computing, 2021
  - ISC'23 tutorial session

Wi4MPI: current support and (known) limitations

- support:
  - x86 & ARM architectures
  - GNU/Linux & *BSD (tested on FreeBSD)
  - C & Fortran
  - MPI 3.1
- limitations:
  - dynamic linking mandatory
  - avoid (or circumvent) `RPATH` [1]
  - `timeout` feature not supported on FreeBSD[2]
  - translation of `MPI_MAX_*` constants for the maximum length of some strings may result in truncation
  - `MPIX_*` [3] dealt with on a case-by-case basis

---

[1] `RPATH` to `RUNPATH` conversion using `chrpath -c` can be a solution

[2] `timeout` feature allows to add a timeout to any MPI function with `WI4_timeout=` environment variables

[3] `MPI_*` functions are experimental functions, in general that will be included in the next version of the MPI norm, but already implemented

4. Applying dynamic translation to key use cases in HPC

Working around limitations of an MPI library

Use case:

- running a GROMACS version compiled against MPICH
- on the target cluster, MPICH can run only on GPU, error on CPU:

```
gmx_mpi: src/mpi/misc/gpu.c:90: PMPIX_Query_cuda_support: Assertion 'mpi_errno' failed.
```

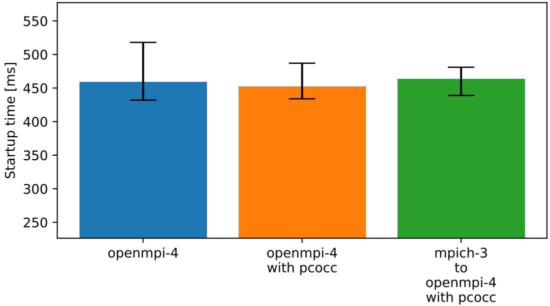
	MPICH	Wi4MPI	Open MPI
Perf. (water GMX50 bench.)	fail	72.99 ns/day	74.23 ns/day

Enabling fast & portable containers

Use case:

- MPICH-based container embedding OSU microbenchmarks
- comparison on 2 AMD Milan nodes at TGCC:
  - Open MPI directly on the cluster
  - Open MPI within a container
  - MPICH within a container to Open MPI using Wi4MPI

`MPI_Init`

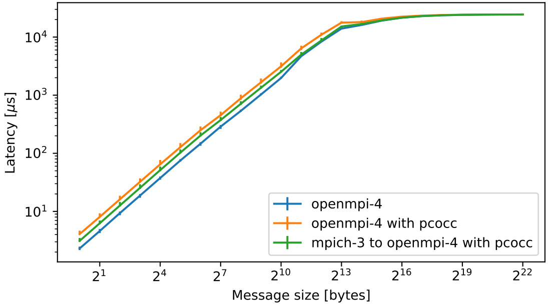


Enabling fast & portable containers

Use case:

- MPICH-based container embedding OSU microbenchmarks
- comparison on 2 AMD Milan nodes at TGCC:
  - Open MPI directly on the cluster
  - Open MPI within a container
  - MPICH within a container to Open MPI using Wi4MPI

`MPI_bibw`

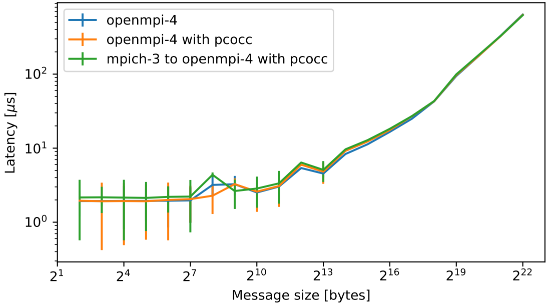


Enabling fast & portable containers

Use case:

- MPICH-based container embedding OSU microbenchmarks
- comparison on 2 AMD Milan nodes at TGCC:
  - Open MPI directly on the cluster
  - Open MPI within a container
  - MPICH within a container to Open MPI using Wi4MPI

`MPI_allreduce`



5. Conclusions

End game? Standardizing the ABI layer

- The MPI Forum is *very likely* to define a C ABI
  - see Hammond et al. 2023 (`https://dl.acm.org/doi/10.1145/3615318.3615319`)
  - convergence is expected; nowadays 2 ABIs cover over 90% of HPC platforms
- Plan is a single-feature ABI-only release for MPI 4.2
  - probably for SC'24
- There is already a prototype available in MPICH
  - `https://github.com/jeffhammond/mukautuva`
- More info at the MPI ABI Working group:
  - `https://github.com/mpiwg-abi`
- Wi4MPI is cited as reference implementation

Wi4MPI in a nutshell

- **Wi4MPI allows to switch between MPI libraries**
  - it allows greater portability and flexibility of HPC applications, including containerized app
- **Wi4MPI usage is mostly transparent**
  - no significant overhead in most cases studied so far
- **Wi4MPI is still evolving**
  - MPI-4 support
  - Mukautuva (MUK) ABI support
  - *your future contribution?*

Thank you for your attention!

And many thanks to the Wi4MPI team (in 2024):

- CEA: Bruno Frogé, Marc Joos, Marc Pérache
- LLNL: Nathan Hanford, Edgar Leon
- Eolen/AS+: Adrien Cotte, Lydéric Debusschaert, Vincent Ducrot, Kevin Juilly, Guillaume Lescroart

and all the people who have contributed to the project since 2016

CEA
Centre de Bruyères-le-Châtel | 91297 Arpajon Cedex
T. +33 (0)1 69 26 40 00 | F. +33 (0)1 69 26 40 00
Établissement public à caractère industriel et commercial
RCS Paris B 775 685 019