Open MPI on Piz Daint

Technical Report

Forschungszentrum Jülich
Institute for Advanced Simulation
Jülich Supercomputing Centre
High Performance Computing in Neuroscience Division
Neuroscience Simulation and Data Laboratory
Multiscale Simulation and Design Team

Contents

1. Introduction

The Fenix Infrastructure aims to align the computing and storage services of six European supercomputing centres. The Interactive Computing E-Infrastructure (ICEI) is the first version of the mentioned infrastructure, being this part of the European Human Brain Project (HBP) [1] [2]. Therefore, Piz Daint, one of the systems of the Swiss National Supercomputing Centre (CSCS), is at the disposal for conducting brain research simulation as a part of the HPB project.

Precisely to conduct brain research simulations on the CSCS's facilities, the Piz Daint system must provide a MPI implementation supporting the Socket-style communication approach, since such approach has been embraced as the mechanism for data communication and components coordination of the Multi-Scale Co-Simulation Framework.

The Co-Simulation Framework, developed by the Jülich Supercomputing Centre (JSC), enable researchers to conduct research by using different brain simulator which use and produce data in different format and time scale. This is the reason why the Socket-style communication is key for the proper operation of the Multi-Scale Co-Simulation Framework.

In this technical report is presented different approach to make Open MPI 4.1.1 work on Piz Daint. In the section 2 is briefly described how MPI interact with SLURM and PMI frameworks. The section 3 details the followed compilation process and the outcomes produced when the Socket-style test were performed.

2. Background

2.1. Rationale

The Co-Simulation Framework uses the MPI's Socket-style Communication approach [3] to synchronize the components and data transfer, hence the supercomputing system where the Co-Simulation Framework is going to be used, must provide a MPI framework implementation supporting the mentioned communication mechanism.

2.2. Piz Daint

The Piz Daint is a hybrid Cray XC40/XC50 supercomputing system on the Swiss National Supercomputing Centre (CSCS) [4]. As a member of the Fenix initiative, the CSCS infrastructure can be used for conducting research of the European Human Brain Project (HBP) [1].

2.3. Open MPI 4.1.1

Open MPI version 4.1.1. implements the Socket-style communication as required by the Co-Simulation Framework, nevertheless as is depicted in Figure 1, *Open MPI* is implemented by using a layered scheme, which implies that the *OPAL* layer is intricately linked to the infrastructure where *Open MPI* is being compiled or used.

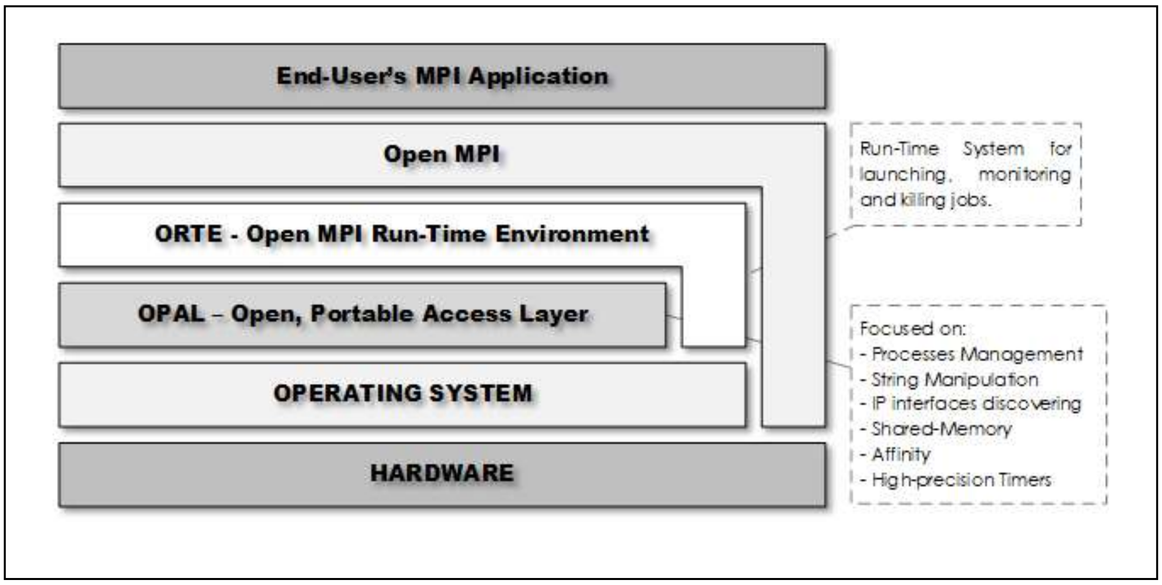
End-User's MPI Application
Open MPI
ORTE - Open MPI Run-Time Environment
OPAL – Open, Portable Access Layer
OPERATING SYSTEM
HARDWARE
Run-Time System for launching, monitoring and killing jobs.
Focused on:
- Processes Management
- String Manipulation
- IP interfaces discovering
- Shared-Memory
- Affinity
- High-precision Timers

Figure 1. MPI Main Abstraction Layers [7] [8]

2.4. PMI

The workload manager used on Piz Daint is SLURM wich provides the PMI library, wich represent the OPAL interface used by Open-MPI for managing the processes and Socket-style communication as is depicted in the **Error! R**

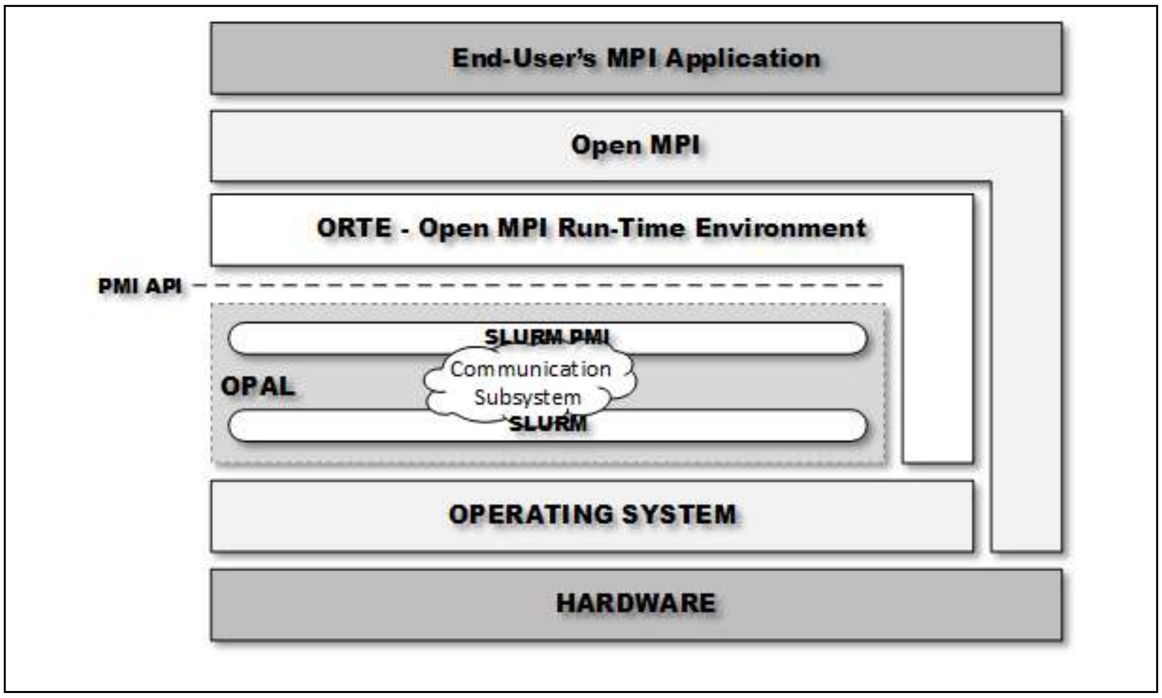

Figure 2. SLURM PMI as OPAL Layer [9]

eference source not found..

The PMI libs found on Piz Daint are the followings:

```
@daint104:~> find /opt -name libpmi.so
/opt/cray/pe/pmi/5.0.11/lib64/libpmi.so
/opt/cray/pe/pmi/5.0.16/lib64/libpmi.so
/opt/cray/pe/pmi/5.0.17/lib64/libpmi.so
```

Nevertheless, the API provided on the listed libraries, does not provided some functions expected by Open-MPI as is detailed in the following sections.

Apart of the inconvenient founds during run-time when PMI is used by Open-MPI, there are two important considerations to ponder [5], namely:

> *"**NOTE**: OpenMPI has a limitation that does not support calls to MPI_Comm_spawn() from within a Slurm allocation. If you need to use the MPI_Comm_spawn() function you will need to use another MPI implementation combined with PMI-2 since PMIx doesn't support it either."*
>
> *"**NOTE**: Some kernels and system configurations have resulted in a locked memory too small for proper OpemMPI functionality, resulting in application failure with a segmentation fault. This may be fixed by configuring the slurmd daemon to execute with a larger limit. For example, add "LimitMEMLOCK=infinity" to your slurmd.service file."*

2.5. PMIx

The aim of PMIx is to solve the launch time scaling by doing the following approach [6]:

- Pre-loading known information into Resource Manager/Scheduler
- Pre-assigning communication end-points.
- Reducing data exchange during INIT
- Orchestrating launching procedure.

The similar way as PMI, Open-MPI uses PMIx by means of an API. Evidently, the workload manager must provide de support for such libraries as is shown in the Figure 3. The restrictions are related to which processes manager LIBs have been attached to SLURM in the case of Piz Daint, which should be referenced/used by the Open-MPI compilation process.

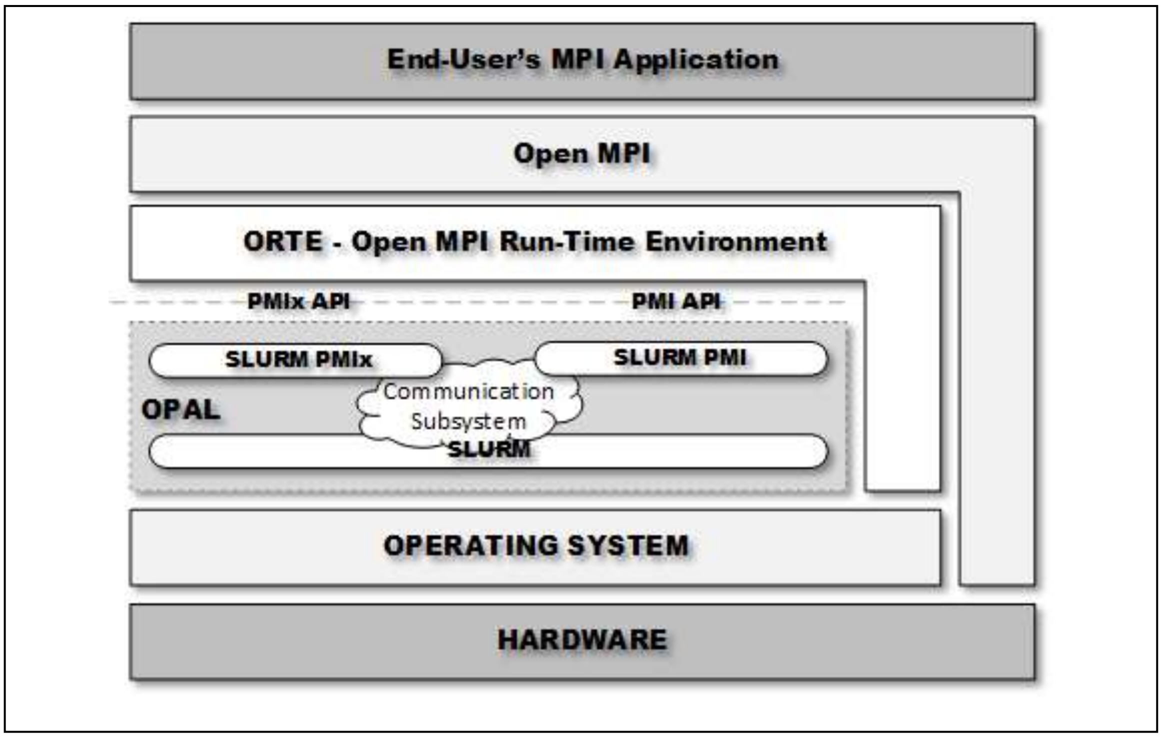

Figure 3- PMIx [6]

Particularly for PMIx, the SLURM documentation makes the following notes [5]:

> "***NOTE***: *Since both Slurm and PMIx provide libpmi[2].so libraries, we recommend to install both pieces of software in different locations. Otherwise, both libraries provided by Slurm and PMIx might end up being installed under standard locations like /usr/lib64 and the package manager erroring out and reporting the conflict.*"

> "***NOTE***: *If you are setting up a test environment using multiple-slurmd, the TmpFS option in your slurm.conf needs to be specified and the number of directory paths created needs to equal the number of nodes. These directories are used by the Slurm PMIx plugin to create temporal files and/or UNIX sockets.*"

3. Compilation Procedure

3.1. Prerequisites

3.1.1. SLURM’s – libpmi.so and libpmi2.so

In order to get access to libpmi.so and lipmi2.so binary libraries, the SLURM’s source code must be used. The SLURM’s version installed on Piz Daint is:

```
@daint103:~> sinfo -V


slurm 20.11.7
```

The source code for such SLURM’s version can be gotten by using the link:

```
https://download.schedmd.com/slurm/slurm-20.11.7.tar.bz2
```

After having compiled the SLURM source code, the lipmi.so and libpmi2.so can be generated by performing:

```
cd <SLURM_BUILD_LOCATION>/contribs/pmi[2]
make -j install
e.g.:
@daint103:.> cd $SCRATCH/build/slurm-20.11.7/contribs/pmi
@daint103:.> make -j install
@daint103:.> cd $SCRATCH/build/slurm-20.11.7/contribs/pmi2
@daint103:.> make -j install
```

3.1.2. PMIx

The procedure how to make SLURM and PMIx is described on [5]. Notwithstanding, the aim is to get the libpmix.so library in order to get Open-MPI compiled referencing such LIB. The steps to get libpmix.so is as follows:

```
wget
https://github.com/openpmix/openpmix/archive/refs/tags/v4.0.0.tar.gz
./autogen.pl
./configure –prefix=<location> --disable-man-pages
Make -j install
```

3.2. OpenMPI 4.1.1

In the following sections the usage of different configuration flags are shown in order test OpenMPI using different scenarios. Nevertheless, which is common is the OpenMPI source code which can be gotten from:

```
https://download.open-mpi.org/release/open-mpi/v4.1/openmpi-4.1.1.tar.gz
```

4. Test Cases.

4.1. Case 1 – Without referencing expressly any LIB.

Case Description:

Configuring OMPI with no reference to external LIBS, the LIBS found on the system are used.

OMPI Configuration parameters:

```
./configure --prefix=${SCRATCH}/usr --with-slurm
```

OMPI Compilation and Installation:

```
make -j install
```

Test Code Compilation:

```
${SCRATCH}/usr/bin/mpicc haskell_mpi_server.c -o server
```

SLURM reservation:

```
salloc -vv --account=ich028 --partition=debug --constraint=mc --nodes=1
```

Running Test Code:

```
srun --nodes=1 --nodelist=nid00448 --ntasks=1 –exclusive \
--mpi=pmi2 \
./server ThePort
```

Issue Report:

```
--------------------------------------------------------------------------
Your application has invoked an MPI function that is not supported in
this environment.

  MPI function: MPI_Comm_accept
  Reason:       Underlying runtime environment does not support accept/connect functionality
--------------------------------------------------------------------------
[nid00448:26410] *** An error occurred in MPI_Comm_accept
[nid00448:26410] *** reported by process [1977679872,0]
[nid00448:26410] *** on communicator MPI_COMM_WORLD
[nid00448:26410] *** MPI_ERR_INTERN: internal error
[nid00448:26410] *** MPI_ERRORS_ARE_FATAL (processes in this communicator will now abort,
[nid00448:26410] ***    and potentially your MPI job)
aborting job:
N/A
srun: error: nid00448: task 0: Exited with exit code 255
srun: launch/slurm: _step_signal: Terminating StepId=34436577.0
```

4.2. Case 2 – Specifying the Piz Daint's Cray's PMI LIB.

Case Description:

Compiling OMPI by using the reference to the PMI LIBS already installed on Piz Daint.

OMPConfiguration parameters:

```
./configure --prefix=${SCRATCH}/usr \
--with-pmi=/opt/cray/pe/pmi/5.0.17 \
--with-pmi-libdir=/opt/cray/pe/pmi/5.0.17/lib64 \
--with-slurm
```

OMPI Compilation and Installation:

```
make -j install
```

Test Code Compilation:

```
${SCRATCH}/usr/bin/server.c -o server -L/opt/cray/pe/pmi/5.0.17/lib64/ -lpmi
```

SLURM reservation:

```
salloc -vv --account=ich028 --partition=debug --constraint=mc --nodes=1
```

Running Test Code:

```
srun --nodes=1 --nodelist=nid00008 --ntasks=1 --exclusive ./server ThePort
```

Issue Report:

```
--------------------------------------------------------------------------
Your application has invoked an MPI function that is not supported in
this environment.

  MPI function: MPI_Comm_accept
  Reason:       Underlying runtime environment does not support accept/connect functionality
--------------------------------------------------------------------------
[nid00008:69628] *** An error occurred in MPI_Comm_accept
[nid00008:69628] *** reported by process [2953969664,0]
[nid00008:69628] *** on communicator MPI_COMM_WORLD
[nid00008:69628] *** MPI_ERR_INTERN: internal error
[nid00008:69628] *** MPI_ERRORS_ARE_FATAL (processes in this communicator will now abort,
[nid00008:69628] ***    and potentially your MPI job)
aborting job:
N/A
srun: error: nid00008: task 0: Exited with exit code 255
srun: launch/slurm: _step_signal: Terminating StepId=34451474.1
```

4.3. Case 3 – Using SLURM's PMI LIB.

Case Description:

Since the SLURM installation on Piz Daint does not provide the PMI LIBS, the SLURM sources should be used to generate such LIBS in order to be used by the OMPI configuration/compilation process

NOTE: libevent.so is required

OMPConfiguration parameters:

```
./configure --prefix=${SCRATCH}/usr \
--with-pmi=${SCRATCH}/opt \
--with-pmi-libdir=${SCRATCH}/opt/lib \
--with-libevent=${SCRATCH}/opt \
--with-libevent-libdir=${SCRATCH}/opt/lib \
--with-slurm
```

OMPI Compilation and Installation:

```
make -j install
```

Test Code Compilation:

```
${SCRATCH}/usr/bin/mpicc server.c -o server -lpmi
```

SLURM reservation:

```
salloc -vv --account=ich028 --partition=debug --constraint=mc --nodes=1
```

Running Test Code:

```
srun --nodes=1 --nodelist=nid00009 --ntasks=1 –exclusive ./server ThePort
```

Issue Report:

```
scratch/snx3000/ringlsch/testing_code/./server: symbol lookup error:
/scratch/snx3000/ringlsch/usr/lib/openmpi/mca_pmix_cray.so: undefined symbol: PMI2_Initialized
srun: error: nid00009: task 0: Exited with exit code 127
srun: launch/slurm: _step_signal: Terminating StepId=34453122.2
```

4.4. Case 4 – Using PMI2 from SLURM’s source code.

Case Description:

Testing the usage of `libpmi2.so` as PMI interface.

NOTE: The OMPI’s configuration process is the same of the CASE 3, nevertheless, during the compilation of user’s test code, the `-lpmi2` is used.

OMPConfiguration parameters:

```
./configure --prefix=${SCRATCH}/usr \
--with-pmi=${SCRATCH}/opt \
--with-pmi-libdir=${SCRATCH}/opt/lib \
--with-libevent=${SCRATCH}/opt \
--with-libevent-libdir=${SCRATCH}/opt/lib \
--with-slurm
```

OMPI Compilation and Installation:

```
make -j install
```

Test Code Compilation:

```
${SCRATCH}/usr/bin/mpicc server.c -o server -lpmi2
```

SLURM reservation:

```
salloc -vv --account=ich028 --partition=debug --constraint=mc --nodes=1
```

Running Test Code:

```
srun --nodes=1 --nodelist=nid00008 --ntasks=1 --exclusive ./server ThePort
```

Issue Report:

```
/scratch/snx3000/ringlsch/testing_code/./server: symbol lookup error:
/scratch/snx3000/ringlsch/usr/lib/openmpi/mca_pmix_cray.so: undefined symbol: PMI_Get_version_info
srun: error: nid00008: task 0: Exited with exit code 127
srun: launch/slurm: _step_signal: Terminating StepId=34453719.1
```

4.5. Case 5 – PMIx LIB.

Case Description:

Referencing PMIx LIB.

NOTE:

OMPConfiguration parameters:

```
./configure --prefix=${SCRATCH}/usr \
--with-pmi=${SCRATCH}/opt \
--with-pmi-libdir=${SCRATCH}/opt/lib \
--with-libevent=${SCRATCH}/opt \
--with-libevent-libdir=${SCRATCH}/opt/lib \
--with-pmix=${SCRATCH}/opt \
--with-pmix-libdir=${SCRATCH} \
--with-slurm
```

OMPI Compilation and Installation:

```
make -j install
```

Test Code Compilation:

```
${SCRATCH}/usr/bin/mpicc server.c -o server -lpmix
```

SLURM reservation:

```
salloc -vv --account=ich028 --partition=debug --constraint=mc --nodes=1
```

Running Test Code:

```
srun --nodes=1 --nodelist=nid00008 --ntasks=1 --exclusive ./server ThePort
```

Issue Report:

```
/scratch/snx3000/ringlsch/testing_code/./server: symbol lookup error:
/scratch/snx3000/ringlsch/usr/lib/openmpi/mca_pmix_cray.so: undefined symbol: PMI2_Initialized
srun: error: nid00008: task 0: Exited with exit code 127
srun: launch/slurm: _step_signal: Terminating StepId=34456106.0
```

4.6. Case 6 – Avoiding CRAY and FLUX LIBS installed on Piz Daint.

Case Description:

Compiling OMPI without cray-pmi and flux-pmi support.

OMPConfiguration parameters:

```
./configure --prefix=${SCRATCH}/usr --with-pmi=${SCRATCH}/opt \
--with-pmi-libdir=${SCRATCH}/opt/lib \
--with-libevent=${SCRATCH}/opt \
--with-libevent-libdir=${SCRATCH}/opt/lib \
--with-pmix=${SCRATCH}/opt \
--with-pmix-libdir=${SCRATCH} \
--without-cray-pmi --without-flux-pmi \
--with-slurm
```

OMPI Compilation and Installation:

```
make -j install
```

Test Code Compilation:

```
${SCRATCH}/usr/bin/mpicc haskell_mpi_server.c -o server -l<pmi|pmi2|pmix>
```

SLURM reservation:

```
salloc -vv --account=ich028 --partition=debug --constraint=mc --nodes=1
```

Running Test Code:

```
srun [--mpi=pmi2|cray_shasta] --nodes=1 --nodelist=nid00008 \
--ntasks=1 \
--exclusive ./server ThePort
```

Issue Report:

```
--------------------------------------------------------------------------
A requested component was not found, or was unable to be opened.  This
means that this component is either not installed or is unable to be
used on your system (e.g., sometimes this means that shared libraries
that the component requires are unable to be found/loaded).  Note that
Open MPI stopped checking at the first component that it did not find.

Host:      nid00008
Framework: pmix
Component: cray
--------------------------------------------------------------------------
[nid00008:12094] [[INVALID],INVALID] ORTE_ERROR_LOG: Not found in file ess_pmi_module.c at line
131
--------------------------------------------------------------------------
It looks like orte_init failed for some reason; your parallel process is
likely to abort.  There are many reasons that a parallel process can
fail during orte_init; some of which are due to configuration or
environment problems.  This failure appears to be an internal failure;
here's some additional information (which may only be relevant to an
Open MPI developer):

  pmix init failed
  --> Returned value Not found (-13) instead of ORTE_SUCCESS
--------------------------------------------------------------------------
--------------------------------------------------------------------------
It looks like orte_init failed for some reason; your parallel process is
likely to abort.  There are many reasons that a parallel process can
fail during orte_init; some of which are due to configuration or
environment problems.  This failure appears to be an internal failure;
here's some additional information (which may only be relevant to an
Open MPI developer):

  orte_ess_init failed
  --> Returned value Not found (-13) instead of ORTE_SUCCESS
--------------------------------------------------------------------------
--------------------------------------------------------------------------
```

```
It looks like MPI_INIT failed for some reason; your parallel process is
likely to abort.  There are many reasons that a parallel process can
fail during MPI_INIT; some of which are due to configuration or environment
problems.  This failure appears to be an internal failure; here's some
additional information (which may only be relevant to an Open MPI
developer):

  ompi_mpi_init: ompi_rte_init failed
  --> Returned "Not found" (-13) instead of "Success" (0)
--------------------------------------------------------------------------
*** An error occurred in MPI_Init
*** on a NULL communicator
*** MPI_ERRORS_ARE_FATAL (processes in this communicator will now abort,
***    and potentially your MPI job)
[nid00008:12094] Local abort before MPI_INIT completed completed successfully, but am not able
to aggregate error messages, and not able to guarantee that all other processes were killed!
srun: error: nid00008: task 0: Exited with exit code 1
srun: launch/slurm: _step_signal: Terminating StepId=34474180.2
```

References

[1] "About FENIX," [Online]. Available: https://fenix-ri.eu/about-fenix. [Accessed 18 October 2021].

[2] "Nine Projects from HBP enabled by FENIX consortium partner ETH Zuerich/CSCS e-infrastructure," Human Brain Project, 20 May 2019. [Online]. Available: https://www.humanbrainproject.eu/en/follow-hbp/news/nine-projects-from-hbp-enabled-by-fenix-consortium-partner-eth-zuerich-cscs-e-infrastructure/. [Accessed 18 October 2021].

[3] V. Eijkhout, "MPI topic: Process management," Texas Advanced Computing Center, [Online]. Available: https://pages.tacc.utexas.edu/~eijkhout/pcse/html/mpi-proc.html. [Accessed 18 10 2021].

[4] E. Zürich, "PIZ DAINT," Swiss National Supercomputing Centre, [Online]. Available: https://www.cscs.ch/computers/piz-daint/. [Accessed 18 October 2018].

[5] "SLURM workload manager - MPI and UPC Users Guide," SchedMD, 29 June 2021. [Online]. Available: https://slurm.schedmd.com/mpi_guide.html#open_mpi. [Accessed 24 October 2021].

[6] J. M. S. Ralph H. Castain, "The ABCs of Open MPI," 23 June 2020. [Online]. Available: https://www.open-mpi.org/video/general/easybuild_tech_talks_01_OpenMPI_part1_20200623.pdf. [Accessed 26 October 2021].

[7] "Open MPI China MCP," Texas Instruments, [Online]. Available: https://slidetodoc.com/open-mpi-china-mcp-1-agenda-mpi-overview/. [Accessed 18 10 2021].

[8] P. Carmine Spagnuolo, "The OpenMPI Architecture," Department of Computer Science, Università degli Studi di Salerno, 17 March 2020. [Online]. Available: https://www.codingame.com/playgrounds/47058/have-fun-with-mpi-in-c/the-openmpi-architecture. [Accessed 18 October 2021].

[9] P. Balaji, D. Buntinas, D. Goodell, W. Gropp, J. Krishna, E. Lusk and R. Thakur, "PMI: A Scalable Parallel Process-Management Interface for Extreme-Scale Systems," in *7th European MPI users' group meeting conference on Recent advances in the message passing interface*, Berlin, 2010.